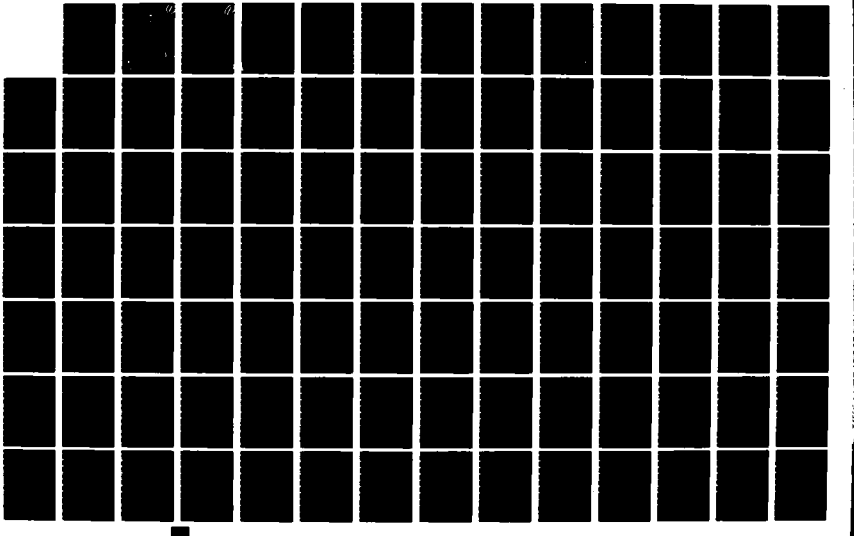
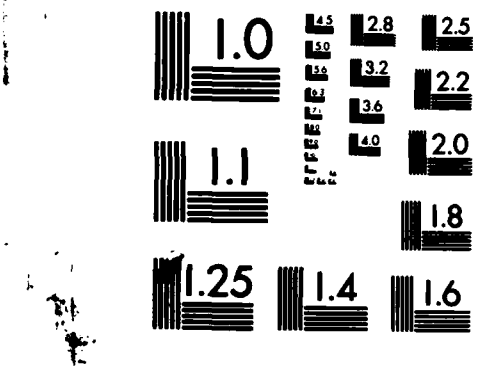
AD-A138 427 PRINTED CIRCUIT BOARD LAYOUT BY MICROCOMPUTER(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING E W KRAUSMAN DEC 83 AFIT/GE/EE/83D-35 1/3

UNCLASSIFIED F/G 9/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A138427

PRINTED CIRCUIT BOARD LAYOUT BY MICROCOMPUTER

THESIS

AFIT/GE/EE/83D-35 Ernest W. Krausman
Capt USAF

DTIC FILE COPY

DTIC ELECTE FEB 29 1984 S D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

84 02 27 058

AFIT/GE/EE/83D-35

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist | Avail and/or Special	
A/1	

DTIC COPY INSPECTED 2

PRINTED CIRCUIT BOARD LAYOUT BY MICROCOMPUTER

THESIS

AFIT/GE/EE/83D-35 Ernest W. Krausman
Capt USAF

Approved for public release; distribution unlimited

DTIC ELECTE FEB 29 1984 D

AFIT/GE/EE/83D-35

PRINTED CIRCUIT BOARD LAYOUT BY MICROCOMPUTER

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Ernest W. Krausman, B.E.E.
Capt USAF
Graduate Electrical Engineering

December 1983

Approved for public release; distribution unlimited.

Acknowledgements

I would like to thank all of the faculty and staff of the Air Force Institute of Technology who helped in the preparation of this thesis. Special thanks are extended to thesis advisor Lt. Col. Hal Carter and reader Maj. Bill Sutton for spending the time to review the drafts and make suggestions. Mr. Orville Wright provided information about the actual printed circuit board fabrication process and was very helpful.

Contents

Contents

Figures

Abstract

Printed circuit board artwork is usually prepared manually because of the unavailability of Computer-Aided Design tools. This thesis presents the design of a microcomputer based printed circuit board layout system that is easy to use and cheap. Automatic routing and component placement routines will significantly speed up the process.

The design satisfies the following requirements: Microcomputer implementation, portable, algorithm independent, interactive, and user friendly. When fully implemented, a user will be able to select components and a board outline from an automated catalog, enter a schematic diagram, position the components on the board, and completely route the board from a single graphics terminal.

Currently, the user interface and the outer level command processer have been implemented in Pascal. Future versions will be written in C for better portability.

Reproduced from best available copy.

Chapter I

Introduction

Once an electronic circuit design is completed on paper, the designer has to test the circuit. Testing usually involves construction of a proto-board or wire-wrap circuit, which are easy to modify. The next step, once circuit modifications have been made and tested, is construction of the actual working circuit. A printed circuit board is most often used because of its reliability, low cost, and capability to be easily reproduced.

This thesis will present the design of a printed circuit board design environment to be implemented on a micro-computer and used by students and faculty at the Air Force Institute of Technology.

Background

The process of transfering an electronic circuit design to a printed circuit board consumes much time and is error-prone when done manually. Here is a basic outline of the steps required to complete a printed circuit board:

1. Design the circuit - prepare a schematic diagram.
2. Decide where to position the components on the board.
3. Layout all of the connections between components.
4. Prepare the artwork from results of 2 and 3.
5. Produce the actual board from the artwork.

The schematic diagram is used to direct the rest of the process. Often, the person who draws the schematic is not the person who makes the printed circuit board. All necessary information must therefore be contained in the schematic.

The next step is the selection of the actual board outline that the circuit will be placed on. This is usually determined by other parameters such as the equipment with which the circuit will be used. The selected board outline is layed out on transparent mylar sheets using strips of opaque black tape.

Now that the board is defined, the components must be positioned within the available area. This is usually done on paper, because of the amount of trial and error involved. The problem is complicated because many components have multiple logically independent elements. A hex inverter chip, for example, has six independent inverters in one package. On the schematic, these six inverters may be scattered all around, making it difficult to decide where to put the chip. Once positions are decided upon, clear plastic stickers with black opaque pads properly oriented for each component are fastened to the mylar sheet.

The most difficult part of the job comes next: the pads must be connected together properly. If a double sided board is being made, then the draftsman has more flexibility. He must decide which connections to make first, because as each connection is laid out, the remaining ones become harder to find room for. For each connection, each segment must be assigned to a layer where it doesn't interfere with existing wires, because wires cannot cross on the same layer. Wires for each layer are laid out with different color tapes (red and blue) on the same mylar sheet,

or each layer has its own mylar sheet with the pads positioned in the same place on each one. In any case, as the draftsman proceeds, he will probably have to remove some previously laid tape to make room for other connections later on. If component placement is done properly, then the actual layout or routing step becomes easier, but is still tedious. A skilled draftsman can become very good at routing, but it takes a lot of patience and practice.

Now that the mylar sheet or sheets have the completed artwork on it, the pattern must be transferred to the copper layer of the printed circuit board. This is a photographic process which uses the clear mylar sheet as an exposure mask. A chemical is applied to the copper surface of the board that makes it sensitive to light. The pattern on the mylar is then transferred to the light sensitive copper by exposing the board to light through either the mylar itself or a negative of it, depending on the process. Another chemical is then used to selectively etch away the unwanted copper, leaving behind copper only where the tape was on the original mylar sheet. If the board has more than one layer, then each layer must be set in registration with the others so that the pads line up on all layers.

Once the artwork is prepared, as many circuit boards as are desired can be manufactured with little additional expense or effort which is the main attraction of printed circuit board technology.

Problem

At the Air Force Institute of Technology, small to medium size circuits designed by students and faculty are manually laid out on double sided boards by a draftsman. Resources are available through the Avionics Laboratory for multi-layer boards, but the original designer of the circuit doesn't do the layout. Because of the time and effort required to produce a printed circuit board, many projects that would benefit from the use of a printed board do not attempt to use them, and others are delayed because of the limited resources available for printed circuit board production.

What AFIT needs is a way for inexperienced designers (students who have never made a printed circuit board before) to create their own printed circuit board artwork in less time than the current manual system requires. While there are commercially available printed circuit board design systems, they are generally not easy for beginners to use. These systems are also capable of producing large, high density circuit boards for production run environments, capabilities which are not required in most cases at academic institutions. This thesis will present the design of a device-independent, graphics oriented printed circuit board layout environment. This system will be designed specifically for users inexperienced in PCB layout, allowing them to produce high quality finished boards.

There are a wide variety of computers, graphics terminals, and plotters available at AFIT, so the ability to support many combinations of hardware is essential. Device independence will also allow support of future hardware acquisitions and permit implementation by other facilities. Interactive graphics with a friendly user interface will allow the inexperienced user to successfully complete the design of his or her board, freeing the draftsman for other work. Also, if it becomes easy to produce finished boards, more boards may be attempted in the first place.

Scope

Because the algorithms to perform component positioning and routing are so numerous, this project will not attempt to develop new ones or select existing ones for implementation. Rather, the primary focus will center around the design of the user environment, including graphics support and file management routines. This environment will provide a base from which follow-on work can build a fully implemented system. Evaluating the resulting user interface will be a part of this thesis effort, to discover any areas that need further development. While routing and placement algorithm implementation is not the focus of this effort, skeletal routines to test the user interface will be implemented.

Evaluation of Methods

Regardless of the system used, there are certain steps which must be carried out when designing a printed circuit board. These steps are circuit description, component placement, board layout, and fabrication. When integrated into a complete system, the user does not have to be aware of the divisions between them. Even though they may be independently operating programs, the user should think he is using only one program. Several commercial printed circuit layout systems have been described in the literature. [1-11] These are integrated systems which include all of the above steps, but there are several implementation differences. The following section will elaborate on the advantages and disadvantages of the various approaches and identify those best suited to a micro-computer environment.

Circuit Description

Once a schematic diagram of a circuit is drawn, that information must be entered into the computer. This information consists of the components to be used and how they are to be connected. At least four approaches are used in commercial systems: a compiled definition language, hand digitization, fully automatic digitization, and interactive keyboard entry.

Definition language - Three of the systems [1,5,8] require the user to describe the components in a special language. Although easy to implement, this method has

several disadvantages: The user must learn the language before he can use it. Logic errors in the input file will not be obvious unless a plot of the schematic is produced. Even then, pinpointing the location of a stray wire can be difficult. With enough experience, compiled languages can be easy to use, but casual or first time users won't have the time to get that experience.

Digitization - Most of the systems [3,4,9,10] use this method to input schematic data, which requires the use of a digitizing tablet. The stylus is used to locate a component and trace or highlight the desired connections. This is quick and reliable, but the user must know how to operate the digitizer. If the same digitizer were to be used everywhere, this would not be of much concern. However, the availability of a suitable digitizer cannot be assumed.

Automatic digitization - One of the systems [11] uses fully automatic digitization. The schematic is drawn on paper and then scanned by the digitizer. This requires extensive software support to extract the circuit from the digitized pattern. Additionally, the schematic must be originally drawn to a set of specifications so it can be read. From the operator's point of view, this is the simplest way to go, but is too expensive for a small system, and training would be required before schematics could be consistently interpreted correctly.

Interactive via keyboard - The last method [6,7] allows the designer to construct the schematic diagram on the graphics display, using an on-screen menu with cursor selection. The display is a window to the entire schematic. This has the benefit of not requiring any special format for the schematic. Additionally, the hardware requirements are minimal. Graphics displays are more readily available than digitizers, and will be required for the placement and routing phases anyway.

Component selection is handled almost identically in each of the systems - a library of predefined component and board parameters is maintained from which the desired components can be selected. Users may add to the library at any time. The type and amount of information about each component includes physical parameters, electrical parameters, logical parameters, and manufacturing data. Although library size on a micro-computer would be necessarily smaller due to storage limitations, it is important to anticipate future needs for the library and be able to support them.

Component Placement

Once the computer knows what components to use and how they are to be connected, they must be positioned on the board. Various algorithms to perform automatic placement are widely used (pairwise and triplet exchange, force-relaxation migration [6]), but they are not as necessary for small boards. The manual placement of

components, if done with good judgement, should be adequate. However, it must also be easy for the user to change his mind and reposition components. Additionally, tne impact of a specific placement should be made visible before routing is attempted.

Commercial systems use manual placement as a first cut at final placement to speed up the process. The assumption is that the designer has a pretty good feel about what components should be close to each otner, and automatic placers simply optimize this initial placement. Even if an automatic placement algorithm is to be implemented, an interactive manual placer is still required.

Additional help is provided to the user in the form of connection distribution displays [3] which provides a wire density plot along both axes, and other graphical aids like rulers, exact component positions, and prompting for next best location.

Board Layout

Now that the components are positioned on the board, the previously identified connections must be routed. The greatest portion of the literature concerning printed circuit board design automation is devoted to descriptions of various routing strategies. The basic types are the Lee or maze router, channel routers, and line search routers. Each of the strategies is adapted for different circumstances, but it is important to realize that even though they seem very different, the same information about

the components and the board configuration is required by them all, and the final results can be represented in similar forms. The focus of this thesis is the development of a data structure that can support any router, not the router itself.

The commercial systems do not use highly sophisticated routines, but rely on combinations of the Lee router and the line search algorithm. Channel routers are designed more for VLSI routing applications, and none of the systems examined incorporated a channel router.

Automatic routers usually can achieve 80-95% of the connections, while the remainder must be inserted manually. A batch mode router will attempt all connections and report failures when finished. This may mean that the operator has to try a new placement and start again, or manually edit the resulting output file. An interactive approach is much more adaptable, and relies less on a particular router. The commercial systems discussed above all incorporate an interactive routing strategy.

The router to be used should be selectable by the user, as should the amount of routing to be done. For example, after an initial attempt to route the entire board, unfinished connections can be attempted one at a time using different routers. Existing paths must be able to be moved to make room for the failed paths. Because the user must be allowed to move paths, a check must be performed to prevent design rule violations.

There are two approaches to achieve 100% routing of a printed circuit board after an initial attempt with an automatic router:

Manual editing - In this case, unrouted connections are manually filled in by the user. By allowing the user to specify where those connections are to go, the possibility exists for design rule errors to be introduced. Also, the graphics display must be able to present the actual board layout in detail. The systems that use this approach incorporate route optimizations to clean up paths entered by the user.

Interactive re-route - A better approach is to allow the user to delete previously routed paths and invoke the automatic router iteratively. The user can usually identify the congested areas easily. By removing paths that are causing problems and routing them later, more of the board can be routed. Because the router does all the actual connections, design rule violations can be eliminated. Of course, the router must enforce the design rules.

Fabrication

After the board is routed, artwork suitable for fabrication must be generated. Because conventional pen plotters do not have the required edge definition to produce a useable mask, they can only be used for a check plot. A photo-plotter must be used to produce the actual artwork. This can be handled by a post-processor, because no interaction is required from the user. Consequently,

development of the post-processor has no direct bearing on the PCB design process. It can be handled off-line in a batch mode and will not be a consideration for this thesis. The commercial systems also have capabilities to produce tapes for numerically controlled drilling machines and ordering information to keep component stocks up to date, which are also not directly related to this effort.

Layout of Thesis

This chapter has examined some commercial mini-computer based printed circuit board design systems, and evaluated the methods and approaches used. Chapter II will explain the requirements that the fully implemented layout system must satisfy, both in general and specifically. Justification of these requirements will also be presented. Chapter III will describe the overall system design of the printed circuit board design environment, and explain the function of the major subsections. Chapter IV will describe in detail the designs of the sections outlined in chapter III. Chapter V will describe any implementation difficulties and present evaluations of the user interface and the file management routines. The final chapter will provide recommendations for future efforts and present conclusions about the success of the project.

Chapter II

General Requirements

Briefly, AFIT needs an interactive printed circuit board design tool that can be implemented on a micro-computer, is portable, and algorithm independent. This tool must allow the user to describe the circuit in sufficient detail so it can be routed and fabricated.

Any student or faculty member should be able to use almost any computer available and easily obtain a finished printed circuit board starting with an ordinary schematic. This is a very simple statement, but it contains the essential requirements for the printed circuit board design environment to be developed by this thesis.

Although a fully implemented printed circuit board layout system is beyond the scope of this thesis, all of the modules will have to meet certain requirements to be able to communicate with other, and the basic routines that the layout modules will require must be implemented as part of the nucleus.

Micro-computer Implementation

There have been many PCB layout systems implemented, but they require the use of mini-computers. They are meant for commercial use, so the range of boards they must handle encompasses large dense boards with hundreds of integrated circuits. Because the circuits designed at AFIT are modest in complexity and many current micro-computer systems approach the capabilities of earlier mini-computers, it is

practical to implement a PCB design system on a micro. No special hardware will have to be purchased to support it.

The initial target machine for implementation is a 4 MHz Z-80 based machine with 48K bytes of RAM and 400K of floppy disk storage. This configuration should be the minimum system considered for this project.

Portable

Because the available software and hardware configurations of the available computers may differ greatly, the entire system cannot be expected to be implemented identically on all of them. However, all differences should be transparent to the user - a constant user environment will enable users to use different implementations with the confidence that it will work the same on each. As a result, all machine dependent features must be isolated. Assembly language cannot be used except for the lowest level I/O routines, because they will have to be re-written for each specific computer and peripheral device. Disk file management routines cannot rely on the resident operating system for the same reason. Consequently, all access to the outside world must pass through a standardized set of procedures.

The language used to write the machine-independent portions must be available on all of the target machines. FORTRAN IV is almost universally available, but it is difficult to manipulate complex data structures efficiently. Although less standardized than FORTRAN, Pascal posesses

much more powerful data structure manipulation facilities. The difficulties with Pascal center around non-standard I/O processing. By forcing all I/O to be performed through standardized subroutines this can be eliminated as a matter of concern.

Another difficulty with Pascal is the separate compilation of procedures and functions. Most implementations allow it, but it is performed differently on each. Since the installation procedure will only have to be performed once on each computer, this is not as important as it first appears, and is outweighed by the ease of coding and the shorter program developement times afforded by Pascal.

Algorithm Independence

To allow for experimentation with different routing and placing algorithms, the system must be modular in structure. This requires that a well defined intermodule communication protocol be adhered to by all components of the system, and that each module make few assumptions about how other modules do their job.

All modules must differ only in the function they perform - the command syntax should not change, for example. There must be no direct I/O performed by any module - all I/O processing must pass through the predefined lower level I/O routines. This will eliminate possible inconsistencies in I/O handling.

A library of standard components and board outlines must be maintained to prevent duplication of effort. A well maintained and complete catalog from which to select components is preferable to defining them each time they are used, and allows users unfamiliar with the detailed component parameters to use them anyway. This library must contain all the information required by the other modules, so that future enhancements can use existing component data. In other words, the structure of the library must be expandable.

Other data structures used by more than one module must include the interconnection list and the location and identification of components on the board. These must be general enough to allow for future expansion of capabilities, not limited to any specific algorithm or board technology.

Interactive

To be truly interactive, the user must be kept aware of the progress being made. He must be able to interrupt a process, make a few changes, and continue. The best way to show progress when routing a board is graphically - draw each connection on the screen as it is completed. Routing can be completed a single wire at a time, or whole chunks can be attempted at once. By putting the user in the loop, the demands on the router are lessened and micro-computer implementation becomes attainable.

Placement requires a graphic display indicating possible congested areas on the board as each component is positioned. Immediate feedback allows the user to decide which placement is best. Simply drawing straight lines between electrically common points will indicate where routing problems are likely to appear.

The requirements imposed on the graphics display depend on the complexity of the circuit to be drawn. A low resolution display can handle a small, simple board; higher resolution is required to display more complex boards. Scrolling through windows can increase the effective size of the display at the expense of execution time. A compromise is the use of fixed windows, one of which can be viewed at a time. Four windows would allow four times the limit of the display size to be drawn. For example, the area of the display of the target machine is 8 X 5 inches, with a resolution of 13 mils. If a 50 mil routing grid were used, a board smaller than the display area could be viewed all at once, while a 16 X 10 inch board would fit into four quadrants.

Because an ASCII keyboard is universally available as an input device, it will be used as the cursor controller as well as for regular text input. Joysticks, trackballs, mice and assorted other gadgets may be more powerful in specific implementations, but they cannot be relied on to be generally available.

User Environment

The environment that the user interacts with includes the hardware and the command set. While certain aspects of the hardware cannot be held constant, the command set should not vary. A given sequence of instructions should produce the same results on independent computers. As far as the user is concerned, the only differences between two implementations of the system should be cosmetic. Large systems may be able to handle larger boards or work faster, however.

A normal level of familiarity cannot be assumed - items like directory organization, what constitutes a legal filename, how to access files, because these tasks are not constant between computers and aren't directly related to the job at hand, i.e. designing a PCB.

Users must be shielded from operating system errors because of the non-standard way they are handled. If a command requires that a disk file be opened, for example, the results can be different if the file doesn't already exist. A new file may be created, or another logically different file that happens to have the same name may be written over, neither of which was intended by the user. To avoid problems like these, the system must make explicit checks before issuing commends to the operating system to insure that all required files are where they are supposed to be. Non-recoverable errors must be eliminated wherever possible to keep the user out of computer dependent

situations.

Constantly having to refer to various manuals will discourage many casual users, so help should be available where required. The help provided must be specific enough to allow the user to continue or let him know he can't do what he wants to do. A simple prompt should never be displayed without explanation. Either available options or specific questions should be asked with indications of valid response ranges.

Summary of General Requirements

The printed circuit board design system developed by this thesis, when fully implemented, must allow a user to layout, route and fabricate a printed circuit board working from a schematic diagram. Additionally, the following requirements must be satisfied:

Micro-computer implementation: Able to run in a minimum system consisting of an 8 bit processor, 48K bytes of RAM and 400K disk storage.

Portable: Written in standard Pascal using standardized routines for I/O processing. Cannot be dependent on specific hardware, peripherals, or operating system for proper operation.

Algorithm Independent: Modular in structure to allow experimentation with routing and placement algorithms. Data structures must be extensible for future growth.

Interactive: Use a graphic display to visually show the current state of the system to the user, allowing him to intercede when he wishes.

User Friendly: Maintain a constant user environment between machines and shield the user from machine dependent factors. Provide on-line help when requested and explain available options. Encourage casual and first time users.

Specific Requirements

The requirements addressed in the previous section apply to the entire printed circuit board design system. The following section will describe the types of circuits designed at AFIT and outline the specific requirements that must be satisfied by the various pieces of the system. These specific requirements include actual board parameters, circuit complexity, implementation requirements, graphics display capabilities, plotting capabilities, and the user interface.

Board Parameters

In general, the maximum board size that the system will be able to handle will depend on the specific capabilities of the host computer. Rather than leaving this open-ended, however, a practical limit on board size is 12 X 8 inches. This is the largest size board that is commonly designed presently, and most are smaller. Larger boards can be designed in pieces, but the limits of circuit complexity described below will make larger boards impractical.

Double sided boards are the most common variety that are designed by hand, and four layer boards are currently supported by MOSIS. The problem with multiple layers is the amount of additional information that must be maintained. For example, with double sided boards, all vias exist on both layers at the same spot. With more layers, it is possible to have vias between inner layers that don't pass through to the outer layers. This requires via information

to be maintained separately for each layer. To support future advances in technology, the data structures used by the router must recognize the existence of multiple layers, even if not all of the layers are supported. A practical upper limit on the number of layers to be considered is four, with two types of vias: those that penetrate all layers, and those that go between adjacent layers only.

Routers that do not support all the layers will only use the ones they recognize, but support for all four layers must be built into the interface of the router and the other modules. Preferred directions of wire segments on each of the layers must not be constrained by the data structure, because different routers assign segments to layers using other considerations besides horizontal or vertical orientation.

Circuit Complexity

To keep the circuit density reasonable, the number of integrated circuits will have to be limited to about 1 per square inch. This will allow about 90 16 pin IC's on a 12 X 8 inch board, or 50 on a 10 X 5 inch S-100 board. Higher densities would quickly outstrip the memory and disk capacities of most micro-computers. Even these densities may be too high for some systems, but the data structures must still work, i.e. be downward compatible.

A routing grid size of 50 mils will support a maximum of one wire between adjacent pins of an IC. While higher densities are possible, the complexity of the circuits to e

designed by this system do not require higher densities. The current limitation on grid size is a minimum trace width and separation of 10 mils, so a 20 mil grid is the minimum that can be fabricated. Even though a router may support such a small grid, for this project a 50 mil grid will be the minimum size grid required. Gridless routers may make more efficient use of board space but the data structures required to support them are more complex than those that are constrained to a grid. A slight loss in generality is preferable in this case, because implementation will be that much easier.

Computer Specifications

To realistically implement this system will require a computer with the following capabilities: 48K of memory, two floppy disks, and an ASCII keyboard. Additional hardware requirements for the graphics display will be discussed separately.

48K bytes of memory will be required to allow reasonable module sizes to execute. Because the system will be segmented into independently operating programs, with no common memory allocated, each module will have the entire resources of the computer available as it executes. This dictates that any information exchanged between modules must reside on disk.

At least two disk drives (Single Sided Single Density 8" or Single Sided Double Density 5.25") must be available to allow separation of project files and system modules.

Because the modules will be chained together, they must all be available at the same time. One disk will be dedicated to the support of the system software and must be available on-line at all times. If more than two drives are available, then it may be possible for two drives to be dedicated to system software. In any case, it will be necessary for all of the major modules to be accessable without switching disks.

The keyboard requirements are very minimal - the computer must be able to respond to individual keystrokes and the keyboard must generate the standard ASCII character set must. Because many computers have special function keys, the keyboard input support routines must be able to be configured for each implementation.

Operating System Support

The host operating system under which the printed circuit board design environment will be executed must support the following functions: random access files and error trapping. The error trapping must allow a program to detect and correct its own mistakes, if possible. This will prevent confusing the operator with potentially misleading error messages. Random access files with fixed record lengths will be used extensively, and must be available.

Pascal Requirements

The Pascal compiler must support the following capabilities: random access files, separate compilation of procedures and functions, and linking with machine language

subroutines. Additionally, it should conform to the standard outlined in User Manual and Report by Jensen and Wirth. The portable portions of the system will assume conformance with this standard. Compilers or interpreters that do not comply may still be used, but implementation may be more difficult.

Graphics Display

To provide a reasonable representation of the circuit board on the video display, the graphics resolution must be high enough to show the minimum routing grid size. To be displayed in actual size, a 50 mil grid will require a resolution of at least 40 points per inch in both the x and y directions. A square aspect ratio will be used by the graphics routines, so the lowest resolution axis must be used for comparison purposes.

The display must be bit-mapped with the graphics memory available for both reading and writing, so the program will be able to determine the status of any pixel, as well as turn it either on or off. Higher level commands, such as drawing lines, boxes, or circles, will be provided by the system if the graphics display does not support them directly. This support will not be as efficient as hardware support, but will allow 'dumb' graphics displays to be used.

Characters and graphics must be simultaneously displayed, but it is not necessary for the text and graphics to be overlaid. Separate areas for text and graphics will be fine. The only purpose of this requirement is to ensure that any messages that the user must see can be displayed without

erasing the graphics display.

Plotter Output

Because the fabrication artwork requires very high edge definition, which cannot be generated by ordinary pen plotters, the plotter will be used only to produce a checkplot in the initial implementation. This checkplot should be drawn in actual size, so the plotter must have a resolution of at least twice the grid size being used. Even higher resolution is necessary to distinguish the pads and other features, so a working minimum resolution of 100 points/inch is called for. Most plotters have much better resolution than this, so no problems exist with this requirement.

For maximum compatibility, the only plotter features which are required are absolute X,Y addressing with the pen either up or down. The paper size is not important, because the plotting routines will partition the plot as necessary. Smaller paper sizes will require more pieces, however.

To distinguish the different layers on the plot, either multiple colors or stipple patterns may be used. Stipple patterns will require blowups of the area of interest to be seen. As with the graphics display, features such as scaling, circle generation, ect. will be provided by the system if the plotter will not perform them directly.

Printer Output

Dot matrix printers with dot addressable graphics capabilities may also be used to provide a printed copy of the video display. The resolution of the printer graphics should at least match that of the video display, but will more likely be greater. Drawing vertical lines on some dot matrix printers is a problem because of gaps between printed lines. This would produce a hard to read plot and would be unacceptable. The printer must be able to produce continuous lines in both horizontal and vertical directions.

Additional plotting routines will be required to support a dot matrix printer. Both the video display and the plotter may be randomly addressed, in the sense that lines may be drawn between any two points independent of what has already been drawn, or what will be drawn next. A dot matrix printer requires that the entire picture first be drawn either in memory or on disk, because the paper can not be reversed without a great loss of accuracy. For each line to be printed, all of the dots must be known beforehand.

Summary of Specific Requirements

In addition to the general requirements, the following specific requirements must be met to ensure practical implementation:

Board Parameters
Maximum Size of 8 inches by 12 inches
At least 2 layers, maximum of 4
Fixed grid size of at least 50 mils

Circuit Complexity
Upper limit of 1 I.C. per square inch

Computer Specifications
48K of Random Access Memory as bare minimum
Two single sided single density 8" or single sided double density 5.25" disk drives
Standard ASCII keyboard
Graphics Display or Terminal
Plotter Interface

Operating System
Random access files
Report errors to applications program

Pascal
Random access files
Separate compilation of procedures and functions
Linking with machine language subroutines
Conform with User Manual and Report standard

Graphics Display
Bit mapped pixel display
Graphics memory accessible for reading and writing
Resolution of at least 40 points/inch
Simultaneous display of text and graphics

Plotter Output
Absolute X,Y addressing
Pen up, pen down
Resolution of at least 100 points/inch

Printer Graphics
Dot addressable graphics capability
Resolution at least as high as Graphics Display
Continuous vertical line capability

Chapter III

System Design

The PCB layout system will consist of the following major components: command processor, component and board data base, and layout modules (the selecter, connecter, placer, and router). Lower level functions include the disk interface, graphics interface, and text I/O modules, containing all procedures dependent on the hardware. Figure III-1 is a top level block diagram showing how they interact. Appendix B contains the details about the Command Processor files, Appendix C defines the Component and Board Data structure, and Appendix D defines the format of the Project Files.

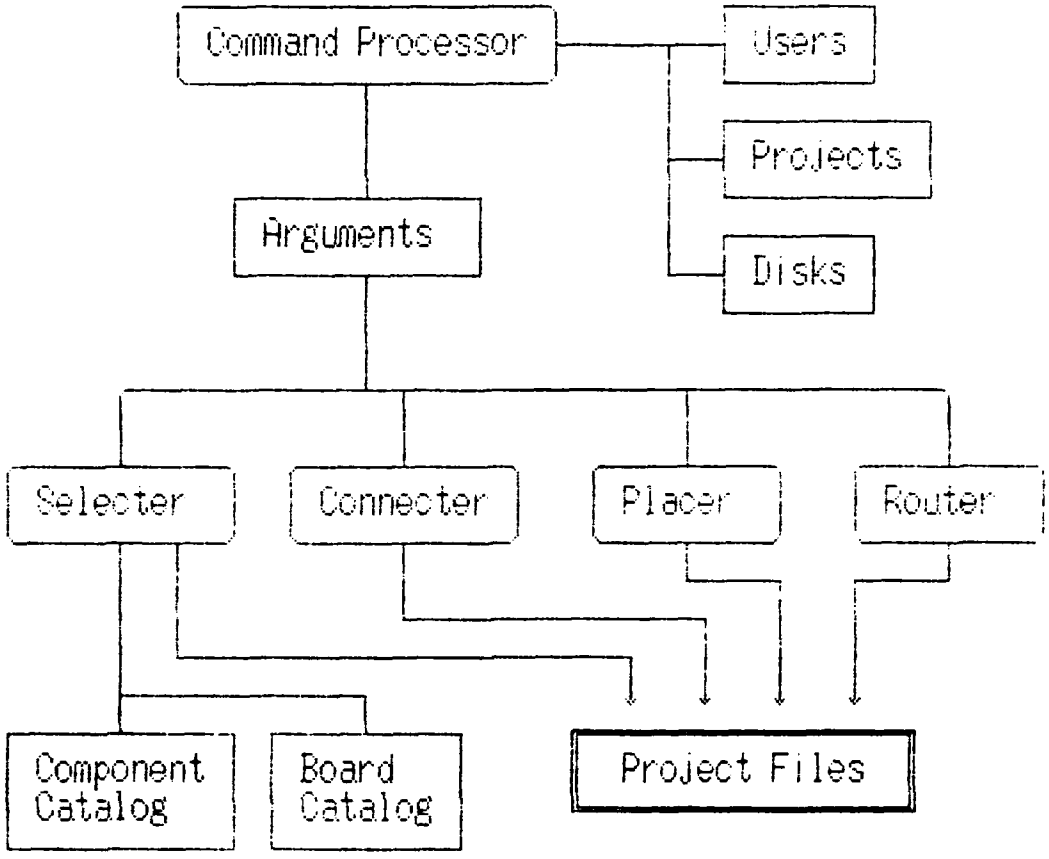


Figure III-1. Overall Block Diagram

Command Processor

Function

This module defines the environment that the designer operates in. Different operating systems perform the same tasks in different ways, causing confusion for operators wnen they switch from one to another. Because the PCB layout must be computer independent, the resident operating system must be made transparent to the user. The Command Processor will basically be a mini-operating system that controls the operation of the computer's resident operating system,

similar to a shell under Unix.

Requirements Satisfied

The burden of user friendliness falls mainly on the command processor. As layout modules are further developed, the only thing the user should notice is an improvement in operation. No new commands will have to be learned, because each module will communicate to the user through the command processor. If the user becomes confused at any point, then help will be available immediately.

A major function of the command processor will be error handling. Operating system error messages are usually difficult to understand, and cannot be relied on to be specific. For example, having the wrong disk in a drive can lead to a wide variety of errors, but the error message reported to the operator probably won't be **'The wrong disk is in drive 1. Please insert disk ABC and press any key to continue.'** General purpose operating systems can't provide nice error messages like that because they don't know what the operator is trying to do.

Algorithm independence is provided by the command processor. The initial layout modules will be very skeletal in function, but the framework will exist into which more complete versions can be inserted.

Structure

The command processor will consist of the following major modules (see figure III-2):

Resolve Error: As mentioned before, the command processor will be responsible for the majority of the error handling. This module will perform this job and return control to the proper module, if possible.

Determine User Identity: This module will be responsible for maintaining an updated list of users and their passwords.

Select Option: Each user will have a list of projects he is currently working on. This module will allow projects to be selected, created, deleted, or worked on. Projects do not have to be completed in one sitting, and this module will be responsible for keeping track of the current state of each project.

Execute Next Module: Once a project is selected to be worked on, the appropriate layout module must be called in. This module is responsible for transferring control to the next module.

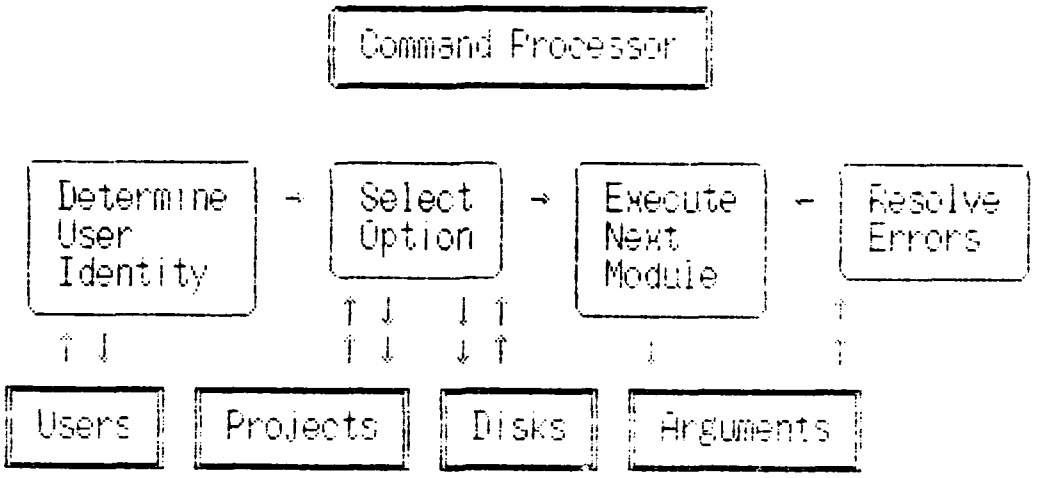


Figure III-2. Command Processor

Because of limited memory space available in many micro-computers, an overlay structure will be implemented. Each module may then consume all memory while executing, with control flow and parameters passed between modules via an argument file and the Command Processor. For example, if the Placer module is finished and the Router is to be executed, the Placer will prepare a command file and load the Command Processor. In turn, the Command Processor will examine the command file and execute the appropriate module, the router in this case.

The only entry point to any module will be through the Command Processor, and all modules will return control back to the Command Processor. This way, each module must only know how to invoke one other module, but the command processor must be able to invoke all other modules.

Selecter

Function

This module will consist of the component and board data files and the library routines required to maintain them. Two levels of information will be maintained - physical and logical. The physical description provides details about the package, such as how many pins there are, how they are spaced, etc., while the logical description assigns names to the pins and to the component. Library maintainence functions will include addition, deletion. and editing of component and board entries as well as the selection of components for projects.

Requirements satisfied

Algorithm independence depends on access to the required information. In other words, no matter what algorithm is to be used for placement or routing, the same information is required by each. The selecter is responsible for maintaining enough information about each component to satisfy the needs of any router. Also, the number of components selected cannot exceed the specified maximum circuit density of 1 I.C. per square inch. If new boards are defined, they must not be allowed to exceed the maximum allowed board size.

Structure

The main modules of the selecter are as follows (see figure III-3):

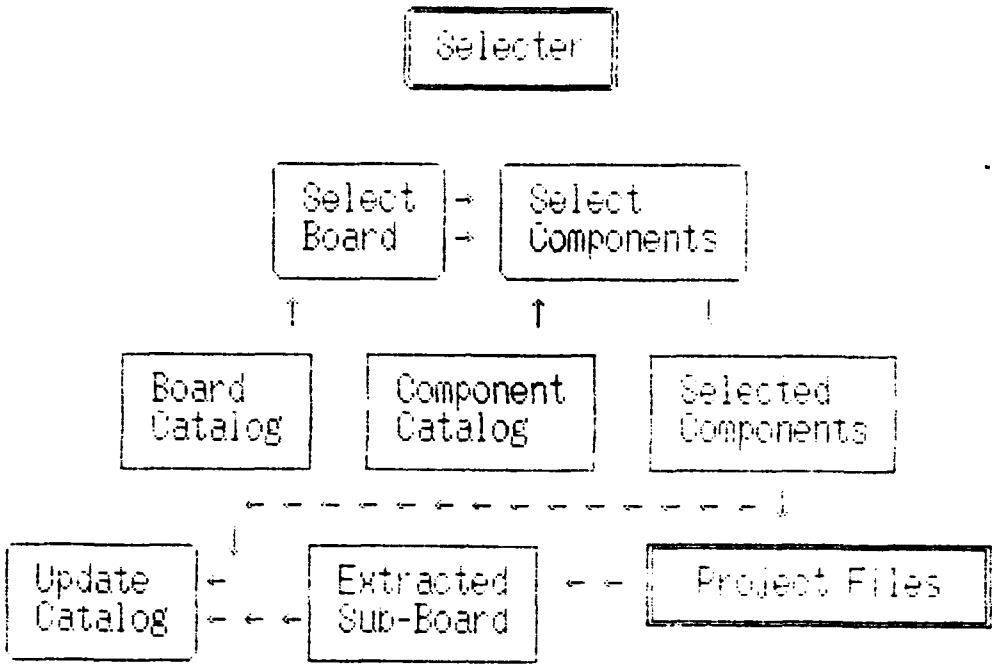


Figure III-3. Selecter

Select Board: The user will be asked to select from the various boards available. If it is not presently in the library, then the user will be asked to describe all necessary parameters.

Select Components: This module is very similar to the previous one, but there are many components to select, and only one board per project. Additionally, the physical description of existing components may be used when defining new components.

Update Catalog: If new components were described for the project, the user may wish to add them to the library permanently. Also, existing definitions may be changed if errors were made in the original entry or if the component actually changes.

If the Selecter is re-entered after a portion of the board has been routed, a portion of the board may be extracted and added to the component catalog. This will essentially be a "super-component", but it will be treated like all other components. Because the component physical definition includes a provision for defining traces, it will be possible to use common circuits on many projects.

This has more implications for other types of circuit routing than printed circuit boards, because the components are fixed in size and number of terminals. For VLSI design, where the "wires" used for routing also compose the circuit elements, it is very important to be able define components in terms of other components.

Interface

The Selecter produces a project file containing all necessary information about the components and the board that the user has selected from the catalog. Each individual project will have its own subset of the master catalog.

Connecter

Function

The Connecter is responsible for determining from the user how the components are to be electrically connected. Each set of electrically common points is referred to as a net, and is usually identified by some signal name.

To allow for maximum routing flexibility, connections will be specified in terms of logical elements, or gates, instead of actual component pins. For example, consider a TTL 7400 Quad NAND. This component has four logically identical two input NAND gates, which are independent of each other. Each gate used in the circuit may be assigned to any one of the available positions within the component.

Requirements Satisfied

While the command processor is responsible for most of the user friendliness, the layout modules must be interactive. In its final form, the connector may include complete schematic entry editing facilities, but the initial implementation will not include this capability. Instead, the user will identify all elements associated with each net. Then for each net, the elements will be drawn on the display with available terminals highlighted. Identification of which highlighted terminals should be connected completes the description of that net.

By only allowing a net to be composed of terminals not associated with any other net, many possible errors can be avoided, which also contributes to a user friendly system. For example, elements like resistors have two terminals which are interchangeable until one side is connected. Once one terminal is assigned to a net, it becomes unavailable for further consideration.

Structure

This module will be simple in structure because the user will be doing all of the work. Each of the modules described below will be an interactive procedure. Here is the general job required of the connecter (see figure III-4):

Net Identification: The user will be asked to identify each signal net by name. For each net, the user will be prompted for the gates or elements which belong to that net. As the elements are identified, they will be displayed on the screen with available terminals highlighted. Once the net elements have been selected, the user will be asked to identify which of the available terminals belong to the net being defined.

Note that the components selected previously will determine the elements which are available to choose from. For example, each 7400 component provides four 2 input NAND elements to the pool. Other components correspond one-to-one with logical elements, such as discrete resistors and capacitors.

Net Editing: After all the nets have been defined by the user, any element with unused terminals will be displayed for verification by the user. At this time, any previously described net may be edited.

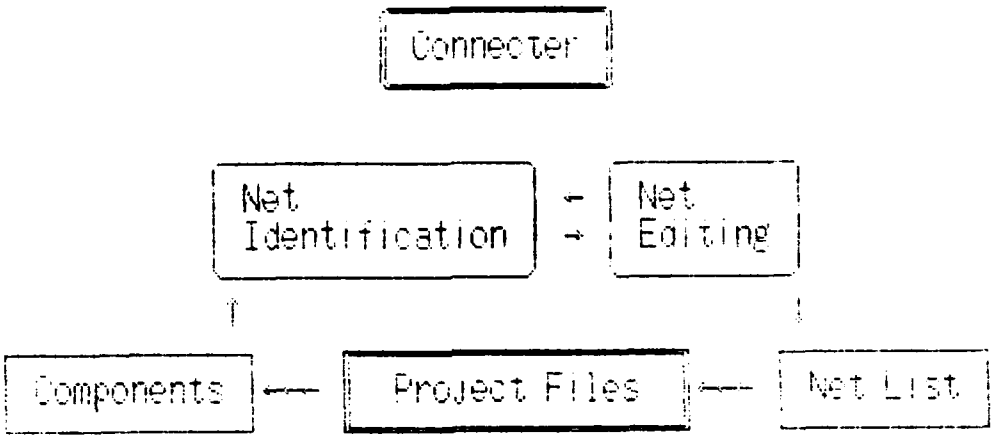


Figure III-4. Connecter

Interface

The project file will be updated to indicate the names of the signal nets and which elements belong to each net. If a component consists of more than one element, then final assignment of elements to components will be handled by the next module, the placer.

Placer

Function

The Placer will decide where to position the components on the board and will assign elements to components in order to make the routing job as easy as possible. Although automatic placing algorithms have been developed, the first implementation of this system will require the user to do the actual placing and gate assignment.

Requirements Satisfied

Like the connecter, the placer must be interactive and user friendly. Interactive graphic displays will allow the user to see the impact of certain placements as he goes along. As each component is positioned on the screen, a line will be drawn from each of its terminals to the closest terminal in the same net already placed. The resulting mass of lines will show possible congested areas and be a guide to further placement. Selection of whether or not to display the lines will be under the control of the user, and can be toggled at will. As components are positioned, the maximum circuit density cannot be exceed. The placer is responsible for enforcing this requirement.

Structure

The component placement problem consists of three subtasks as outlined below (see figure III-5):

Initial Placement: For each net in the project file, the components involved will be displayed on the screen. Each component will be positioned on the board by the user. The nets with the most components will be positioned first so that they can be grouped together.

Element Assignment: When all of the components are in place, the user may wish to rearrange the element-component assignments. This will be easier to visualize once the components are in position.

Placement Improvement: Improvements to placement may be made by exchanging components, either automatically or under user control. Initially, the user will be in total control of the placement process, but future versions may include automatic placement.

The process of reassigning elements may be repeated after the components are rearranged until the user is satisfied with the positions.

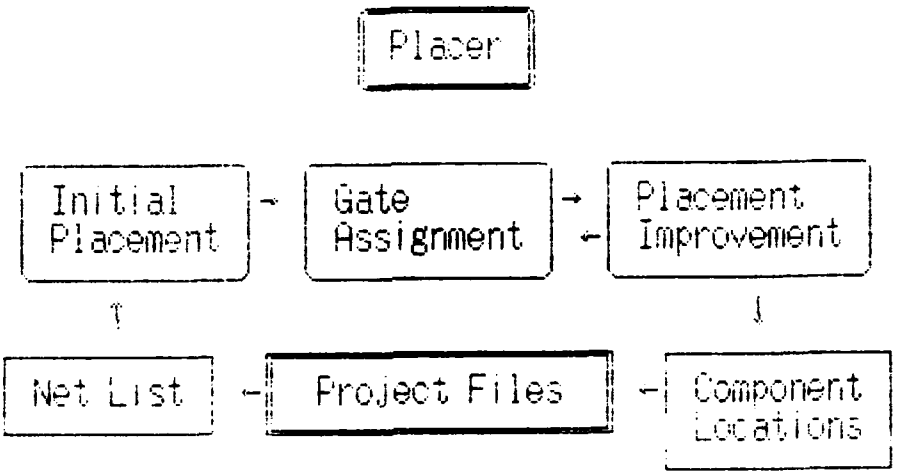


Figure III-5. Placer

Interface

Once the components are positioned, the placer will resolve the references to the net-list in the project file, assigning x y pin coordinates to each element terminal.

Router

Function

The project file now contains the names and terminal locations of each point to be connected together. The router's job is to correctly connect each net without cross-connections with other nets and without leaving any net portion unconnected. Several algorithms have been discussed in the literature, which fall into three general catagories - maze, line, and channel routers. Selection of a routing algorithm is beyond the scope of this thesis, and tne initial implementation will basically be a manual system, in which the user decides where to place connections. However, all of the information required by a routing algorithm is available along with the necessary support routines.

Requirements Satisfied

Experimentation with different routing strategies will be made much simpler with the support provided by this system. The router will be interactive as well. Most of the specific requirements, however, must be satisfied by the router, such as the grid size, and the number of layers.

Structure

There are many ways to partition the routing problem, and here are the most basic steps to be completed (see figure III-6):

From-to determination: There are many ways to connect a given list of terminals together into a net. Usually the closest terminals are connected together, followed by the next closest, until the net is broken down into a set of terminal pairs, often referred to as from-to pairs (FROM terminal A TO terminal B). The initial implementation will rely on the user to do the from-to determinations, so this module will simply ask the user to decompose each net into an equivalent set of terminal pairs.

Net Ordering: Now that the nets are composed of pin pairs, the order in which to route the nets must be determined. Once again, the shortest nets are usually routed first, but some routers ignore net groupings and simply route individual from-to's, while others ignore from-to groupings and route entire nets at a time. In any case, some order must be chosen by the router. Once again, the user will be asked to decide in what order he wishes to route the nets. Help will be available in the form of information about the sizes of the remaining nets.

Routing: With information about specific terminals to be connected, the router is responsible for finding a legal path between those terminals. The initial implementation will require the user to identify where to place traces to connect terminals. The terminals to be connected will be highlighted on the display, and the user will not be allowed to violate any design rules during the routing process.

Plotting: Once a circuit or a portion of a circuit has been routed, the user may want to generate a plot for closer examination. This plot may be drawn to any scale desired, and any section of the board may be isolated. Because the plotting will take much longer than drawing on the screen display, it should only be used when the details omitted from the screen are important.

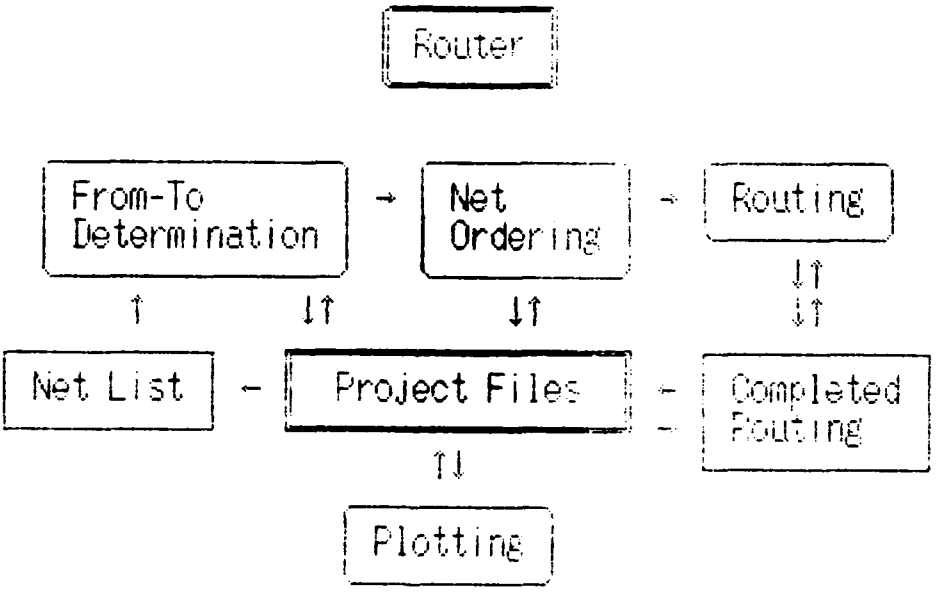


Figure III-6. Router

Interface

All of the information required by the router is present in the project file. As routing proceeds, partially completed results will be maintained in the project file to allow work to be spread out over many sessions if desired.

Graphics Routines

All of the layout modules will require the use of the graphics display. To support them, a set of universal graphics interface procedures will link the portable Pascal modules to the machine dependent hardware.

Two levels of procedures will be written - those supporting basic and advanced features. Basic features include individual pixel control and clearing the screen. Advanced features include drawing lines and boxes. All of the advanced features will be written in portable Pascal and will call the basic routines. If the graphics display hardware supports one or more of the advanced features, the implementer may replace the appropriate subroutine with a machine dependent equivalent that satisfies the calling sequence.

Text I/O

To support the user friendly nature of the system, a query module will provide features such as input validation, error messages, and on-line help. By sending all requests for information through the query module, the format will be identical.

A menu selection module will be available to allow the user to select from among many choices in a consistent manner. As with the query module, all of the menus will have the same structure and commands which will aid the user.

Floppy Disk Management

The entire system cannot be assumed to be available on-line at all times. For example, the program modules, data base files, and project files must be allowed to be spread out over several physically separate floppy disks. Because this will require the user to change disks, the possibility of operator error is greatly increased. To catch these errors and be able to recover from them requires maintainence of a global directory.

The global directory will contain information on the contents of each of the sub-directories. When a file needs to be opened, the global directory will indicate which physical disk must be present. A check can then be made to verify its availability, and prompt the operator if a disk switch is necessary.

In addition to identifying which disk the operator was supposed to insert, the system will be able to tell what disk actually was inserted. Each separate disk will contain an identification file. The disk management routines will ensure that the global directory is kept current to avoid inconsistencies.

Basic file maintainence routines will include creation and deletion of files, and adding and deleting records from files. The files will be composed of fixed length records that may be linked togetner in a list. Access to files may be either direct (by physical record number) or indirect (using the list structure). This will allow variable length

and cnanging data structures for support of the project files and the component and board data.

Chapter IV

Detailed Design

This chapter will describe in further detail the design of the command processor and the text, graphics, and file management routines. The routines form the core of the user interface and were designed to ease implementation of the other modules. The command processor uses the text and file management routines; they will be discussed first, followed by a description of the required graphics routines.

Notation

Different typestyles are used in figures IV-1.0 through -3.15 to identify items as follows:

Lower-case **boldface** is used for variable names

Upper-case **BOLDFACE** is used for File names

Text I/O Routines

Query Modules

The query modules are responsible for obtaining all information from the user that requires numeric or string responses. The Help module is invoked if the user enters a question mark ('?') at any prompt. Editing functions include insertion, deletion. See figures IV-1.0 to -1.4 for more details.

```
Querynum - Accepts numeric responses
           Checks for range
           Converts answer to default units
Querystr - Accepts textual responses
           Checks for length
           Converts answer to upper or lower case
QueryY/N - Accepts Yes or No answers
```

Module Name: Querynum Level: 1

Called by: System Routine available to all modules

Calls: Getnumber

Function: The user will be asked for a numeric answer to a question. The result will be converted to the default units and checked against the minimum and maximum values allowed. If the user wants help, it will be available.

Entry Conditions: **prompt** = Question the user is being asked, **units** = units (mils, inches, mm, none) from which the answer will be converted, **min** and **max** are the limits on a valid response, and **help index** = entry into **HELP FILE.**

Exit Conditions: **answer** will contain the response entered by the user which has been converted to the default units (mils or none). It will lie in the range **min** to **max.**

Pseudo Code:

```
Begin Querynum(prompt,units,min,max,answer,help index)
   Repeat
      Move cursor to querybox
      Display prompt, blank space, units
      Convert min and max to proper units
      Display 'Range:', min ,' to ', max
      Getnumber( answer , help request, unit change )
      If unit change is set
         Then Case units of
              none: do nothing
              mils: units = inches
            inches: units = mm
                mm: units = mils
      If help request is set
         Then Help ( help index )
      Convert answer to units
      Check for range violation
      If there is a range violation
         Then Move cursor to errorbox
              Display 'Response not within specified range'
   Until no(help request or range violation or unit
change)
End.
```

Figure IV-1.0 Numerical Query Function

Module Name: Getnumber Level: 2

Called by: Querynum

Function: The following characters will be accepted from the terminal: the digits 0-9, a decimal point, and a comma. Additionally, the following editing functions will be recognized: Left and Right Arrows will move the cursor, ^S will insert a space at cursor, ^D will delete the character under the cursor, ? will ask for help, ^U will request a change in units, and Return will terminate entry. When Return is pressed, the string of digits will be converted into a number. If a unit change was specified, the unit change flag will be set.

Exit Conditions: **answer** will contain the numeric value of the number string that was entered. **help request** will be set if help was asked for. **unit change** will be set if the units were requested to be changed by the user.

Pseudo Code:

```
Begin Getnumber ( answer, help request, unit change )
   Move cursor to end of prompt
   Repeat
      Get character from keyboard
      Case character of
      '0'-'9': Add character to string
               Echo character
          '.': If point is set
                  Then do nothing
                  Else set point
                       Echo '.'
          ',': Echo ','
           ^S: Insert space in string
           ^D: Delete space in string
    leftarrow: If cursor at first character
               Then do nothing
               Else Move cursor left one character
   rightarrow: If cursor at last character
               Then do nothing
               Else Move cursor right one character
       return: Convert string to answer
          '?': Set help request
   Until ( character = return or '?' or ^U)
End.
```

Figure IV-1.1 Number Editing

Module Name: Querystr Level: 1

Called by: System Routine available to all modules

Calls: Getstring

Function: The user will be asked to enter a text answer in response to a question. The answer will be converted to uppercase or lowercase, or no conversion will be done. The length will be checked against minimum and maximum allowed lengths. If help is requested, it will be provided.

Entry Conditions: **prompt** is the question being asked, **case** will indicate the type of case conversion to be done, **min** and **max** will bound the length of the string, and **help index** will point to the appropriate entry in the **HELP FILE.**

Exit Conditions: **answer** will contain the validated response from the user.

Pseudo Code:

```
Begin Querystr(prompt, case, min, max, answer, help
index)
   Repeat
      Move cursor to querybox
      Display prompt, string of max underlines
      Display 'Response must be between'
               min 'and' max 'characters'
      Getstring ( answer, min, max, help request )
      If help request
         Then Help ( help index )
   Until no help request
   Case case of
      upper: Convert  answer to UPPERCASE
      lower: Convert  answer to lowercase
      none: do nothing
End.
```

Figure IV-1.2 String Query Function

Module Name: Getstring Level: 2

Called by: Querystr

Function: Any printable ASCII character string will be accepted from the terminal. The string will be checked for length before control returns to the caller. Editing functions will be provided, and help will be requested if desired by the user.

Entry Conditions: **min** and **max** contain the bounds of the length allowed for the string.

Exit Conditions: **answer** will contain the response entered by the user and **help request** will be set if the user asked for help.

Pseudo Code:

```
Begin Getstring
   Move  cursor to end of prompt
   Repeat
     Get character from terminal
     Case character of
            ^S: If length of string = max
                   Then do nothing
                   Else Insert space into string
            ^D: If length of string = 0
                   Then do nothing
                   Else Delete character from string
     leftarrow: If cursor at left end
                   Then do nothing
                   Else move cursor left one character
    rightarrow: If cursor at right end
                   Then do nothing
                   Else move cursor right one character
        return: If length of string < min
                   Then character = null
                   Else    answer = string
           '?': Set help request
Printable char: Add character to string
 Anything else: character = null
   Until character = return or '?'
End.
```

Figure IV-1.3 String Editing

Module Name: Queryy/n Level: 1

Called by: System Routine available to all modules

Calls: Getstring

Function: A yes or no response is requested from the user. Anything that starts with a 'y' will be a yes, anything that starts with an 'n' will be a no, and anything else will not be accepted. Help will be available if required.

Entry Conditions: **prompt** = question being asked and **help index** = pointer to **help message** in **HELP FILE.**

Exit Conditions: **answer** will be Yes or No

Pseudo Code:

```
Begin Yesno ( prompt, answer, help index )
   Repeat
     Move cursor to Querybox
     Display prompt
     Display 'Please respond with Yes or No'
     Getstring ( string, 1, 3, help request )
     If help request is set
        Then Help ( help index )
     character = First character of string
     Case character of
        'Y','y': answer = Yes
        'N','n': answer = No
  Anything else: answer = I dunno
    Until answer  = Yes or No and no help request
End.
```

Figure IV-1.4 Boolean Query Function

Menu System

The function Menu reads a structured menu from the MENU FILE and asks the user to make a selection. A global variable identifies the menu which is currently in memory. If the menu to be displayed is the same one already in memory, then it won't be read in again from the disk. If a different menu is requested, then the old one will be disposed of and replaced with the requested one. See figures IV-2.0 and IV-2.1 for more information.

In addition to fixed format menus stored in the MENU FILE, the menu manipulation routines may also be used with variable information, such as the list of the names of the current users' projects. The three procedures Init List, Build List, and Select List are provided to allow the programs to dynamically create menus.

Regardless of the method used to generate the menu or list, the appearance to the user is the same. Each of the choices is listed on the screen, with a moveable cursor to the left. The up and down arrows are used to position the cursor next to the item of interest. Pressing RETURN will select the item, and pressing ? will provide any available help for the item.

The format of the menu (in memory) is as follows:

Item Text	:what the user sees on the screen.
Item Code	:value returned if this item is selected.
Help Index	:Help message index into HELP file.
Next Item	:pointer to the next item of the menu.
Previous Item	:pointer to the previous menu item.
Next Level	:pointer to another menu. If this is not nil and this item is selected, the next level menu will be displayed.

Text I/O Screen Organization

The modules Query and Menu refer to three separate areas of the display: the Querybox, the Menubox, and the Helpbox. These areas may not overlap on the screen. If the cursor is moved to the Helpbox, for example, then whatever is already there will be erased. After the help message has been displayed, then the box will be left empty.

To allow for various terminal screen sizes, the TextIO module contains all information regarding the location and size of the various screen areas. A function, Get Count, returns the number of lines that have been allocated to each area. Cursor positioning is also performed relative the origin of one of the screen areas.

Module Name: Menu Level: 1

Called by: System Routine available to all modules

Calls: Display Menu, Select Menu Item

Function: A menu of choices will be displayed that will fit on the display all at once. A cursor will be moved by the user until the item to be selected is next to the cursor. Return is pressed to make the selection. To allow for large menus, an item may have sub selections. When one of these items is selected, the next level of menu will be displayed. Control will not return to the caller until a terminal menu item is selected. Help will be provided if necessary.

Entry Conditions: **menu index** will point to the location within the **MENU FILE** which defines the menu structure. The global variable **current menu** contains the **menu index** of the last menu displayed.

Exit Conditions: **selection** will contain the code for the selected terminal menu item.

Pseudo Code:

```
Begin Menu ( menu #, selection )
   If current menu <> menu #
      Then Erase old menu from memory
           Transfer menu from MENU FILE into memory
           current menu = menu #
           menu index = First menu item
   Menu Display ( menu index, selection )
End.

Begin Menu Display ( menu index, selection )
   Move cursor to Menubox
   index = menu index
   Repeat
      Display index.selection text
      index = index.next item
   Until index = nil
   Move cursor to first line of menu
   Select Menu Item
End.
```

Figure IV-2.0 Menu Driver Routine

Module Name: Select Menu Item Level: 2

Called by: Menu Display

Calls: Menu Display (recursive call for nested menus)

Function: From the items displayed, the user will select one. If the choice points to more choices, then Menu Display will be called recursively until a terminal item is selected.

Entry Conditions: **menu index** is a pointer to the dynamic list which contains the information about the current menu choices.

Exit Conditions: **selection** will contain the code of the selected terminal node.

Pseudo Code:

```
Begin Select Menu Item
   index = menu index
   Repeat
      Get character from terminal
      Case character of
        uparrow: If index.previous item = nil
                    Then do nothing
                    Else move cursor up one line
                         index = index.previous item
      downarrow: If index.next item = nil
                    Then do nothing
                    Else move cursor down one line
                         index = index.next item
            '?': Help ( help index )
                 Move cursor back to line
         return: If index.next level = nil
                    Then selection = index.item code
                    Else Display Menu ( index.next level,
                                        selection )
                         If selection = none of the choices
                            Then character = null
         escape: selection = none of the choices
   Until ( character = return or escape )
End.
```

Figure IV-2.1 Menu Selection Function

File Management System

Basic File Access - The LinkFile Module

Both sequential and random access files will be required to fully support the layout modules. For stable files, such as the Help and Menu files, purely sequential access is adequate. Other files are dynamic in content, but will still be accessed sequentially most of the time. To allow for ease of sequential list management, a linked-list file format is used. Each linked file has an index field which forms a doubly-linked circular list of allocated records. All records not belonging to the allocated list are collected in another list called the Free list. Pointers to the head of each list are maintained external to the file in the global directory. Direct access to any record is possible, but care must be taken to avoid accessing unallocated records. Appendix A contains a full description of the basic file format.

A global array called Files maintains all the information required to access any file, including the linked list pointers. The current implementation provides slots for ten files. All files are assigned a slot number, and all file activity is referenced to a particular slot. To keep file buffer overhead down, only two files may be silmultaneously open, but the actual assignment of a file slot to a buffer is automatic and transparent to the user.

The following procedures and functions provide full support for both sequential and random access to a file:

InitF: Initialize a slot in the Files array - All information necessary to access the file must be provided, including the physical location of the file and the access mode. Files may be initialized as either linked or unlinked.

ResetF: Set the pointers to access the first record of the file - for linked files, this will be the head of the list, and for unlinked files, the first physical record.

StateF: Return the current values of the pointers

RoomF: Return the number of unallocated records available

ReadF: Read the next sequential logical record, and update pointers. Successive calls to ReadF will sequentially access all the records in the file. If the file is linked, the link field will determine the order of access. Unlinked files are accessed in physical record number order.

WriteF: If the file is linked, Re-write the last record accessed. The pointers are not changed, so successive calls to WriteF will result in the same record being accessed. If the file is not linked, then successive calls will access sequential records.

InsertF: Insert a new record into the file after the current record. This procedure returns the actual record number assigned to the record, so that it may be accessed

directly later. Because of the linked list structure, the physical position of any record will not change once it is created, regardless of the number of insertions or deletions. This is only valid for linked files.

DeleteF: Delete the current record from the list and return it to the free list. Care must be taken when deleting records that are pointed to by other files. This also is only valid for linked files.

CreateF: This procedure makes a new index for a file, and assigns all of the records to the free list. None of the actual file records are altered, but they become essentially inaccessable.

PosF: Position the pointers to any desired record. Care must be taken to avoid accessing records that are part of the free list, if the file is linked. To avoid problems, the record number returned by the InsertF procedure should be saved if random access is desired. This may then be safely used to position the pointers, provided the record has not been deleted.

Global Directory

The global directory keeps track of all files that the printed circuit board layout system knows about. Although simple in structure, it maintains enough information to locate any file. Internal filenumbers used by the layout modules must be paired up with actual system filenames before they can be accessed. The layout modules will only be working with one project at a time, so each internal

filename may refer to many different physical files, one for each project. Here is the structure of a global directory entry:

Module I.D.: Indicates the modules which may refer to this file. Implemented as a boolean array indexed by module name, with a TRUE value if the corresponding module may access the file. Any combination of the flags is ok, meaning that more than one module may refer to a particular file.

File Num: The basic file management system maintains an array of 10 file slots. This field is the slot number the file will use when it is active.

Project I.D.: The unique reference number assigned to the project to which the file belongs. Two special cases are also represented here - a zero value indicates access to the file is independent of the current project, such as a HELP or MENU file. A value of -1 is used for those files not accessed by any project, or to indicate a template entry (see explanation of template entries in Appendix B).

How Many: The number of logical records that have been allocated to this file. When a file is created, this field is used to determine how much disk space is required for the file.

Linked: A boolean flag indicating whetner or not the file is linked.

File Name: A character array containing the system dependent file name used by the operating system to access the file.

Drive: The disk drive into which the diskette containing the file must be inserted. If incompatible disk drives or formats are being used simultaneously, this will prevent the system from prompting for a diskette to be inserted into the wrong drive.

Disk I.D.: The unique reference number assigned to the diskette on which the file resides. This number will be written on the diskette label and recorded on the diskette itself in the identification file described below.

Rec Len: The logical record length is maintained in this field. The current CP/M implementation requires that this be less than or equal 128 bytes. Because random access files are not part of the Pascal standard, this restriction ensures maximum compatability between different compilers.

Recs Avail: This field indicates how much room is left in the file. The value may range from zero, indicating a full file, to How Many, indicating an empty file.

First: The record number of the first logical file entry. All linked files are maintained as a linked list of records, and this field points to the head of the list. A value of -1 indicates an empty file.

Free: The record number of the next unallocated record. The total number of allocated and unallocated records will always equal How Many.

The last seven fields are used to initialize a particular file slot, as described in Appendix A.

DiskID Module

Each diskette will contain a small identification file, which will uniquely identify it with a diskid number. This number will also be written on the label of the diskette so the user will be able to identify it. The global directory will keep track of all files on each disk and the user will be prompted whenever a diskette switch is necessary. The following procedures have been implemented to oversee the disk switching operations:

Whichdisk: A function that returns the number of the disk currently mounted on the indicated drive. If the disk is not labelled (i.e., is not one of the layout system disks) then a value of zero is returned.

Switchdisk: The user will be prompted to switch disks if necessary, after any open files on the current disk have been closed. A check will be made to ensure that the user inserted the proper disk.

NewDisk: A currently unlabelled disk is labelled, and the user is told to physically identify the disk on the label. This disk id number is the only way the system will prompt for disks, so it is important for the number to be recorded correctly!

Argument Module

To provide for communication between the Command Processor and the other layout modules, the ARGUMENT FILE contains any necessary parameters. When control is transferred to another module, the information from the global directory concerning the files required for the current project and module to be executed is loaded into the ARGUMENT FILE. Conversely, when the Command Processor regains control, it loads the contents of the ARGUMENT FILE to update the global directory and state of the current project. The following procedures have been implemented to support these activities:

Read Args: Load the current state of the system as determined by the contents of the ARGUMENT FILE header, and update the global directory for each file entry. (See Appendix B for a description of the ARGUMENT FILE structure.)

Update Header: Save the current state of the system in the ARGUMENT FILE header.

Load Args: Look up all the files required by the next module in the global directory and copy the information to the ARGUMENT FILE.

LoadFile Module

The complimentary routines to the Argument Module are contained in the LoadFile Module. These two routines will be the first and last statements of every subordinate module, completing the parameter passing loop.

Initialize: Reads the system state and parameters passed from the Command Processor. All other system module initialization is also performed here.

Return: Updates the ARGUMENT FILE and returns control to the Command Processor.

Command Processor

The next section details the design of the command processor. Each main portion of the design is preceded with a chart showing the modules to be described with a double line box around it, and the others have single line boxes.

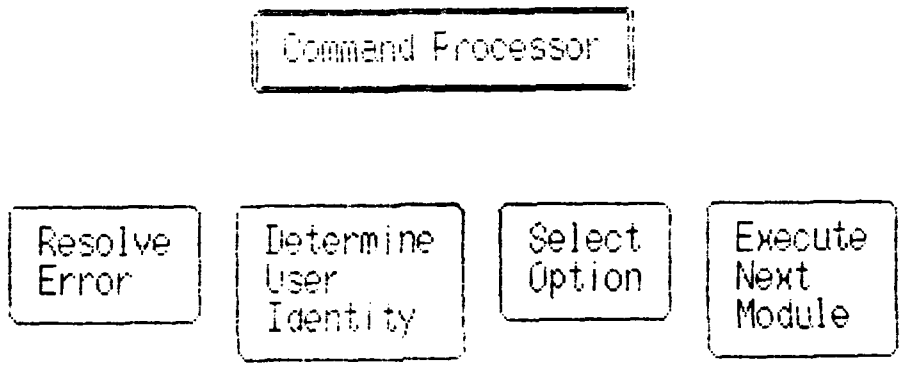


Figure IV-3.0 Command Procedure Routines

Module Name: Command Processor Level: 0

Called by: Initial Entry to System or upon return from Selecter, Connecter, Placer, Router

---

Executes: Selecter, Connecter, Placer, Router

Calls: Determine User Identity, Select Option, Resolve error, Execute next module

---

Function: Controls the Entire PBC design process. Directs the execution of the major layout modules and is the primary interface with the operator.

Entry Conditions: The **ARGUMENT** file contains information about the **current project.**

Exit Conditions: After determining what to do next, the **ARGUMENT** file will be updated and the next major module will be executed.

Pseudo Code:

```
Begin Command Processor
   Look at ARGUMENT file
   Repeat
      If error
         Then Resolve Error
      If current project is Null and no error
         Then Determine User Identity
      If valid user no error
         Then Select Option
      If no error
         Then Execute Next Module
   Until no error
End.
```

Figure IV-3.1 Command Processor Main Routine

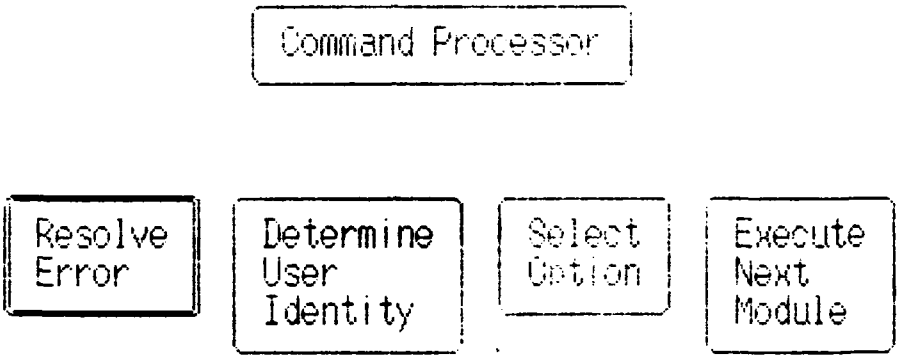


Figure IV-3.2 Resolve Error Modules

Module Name: Resolve Error Level: 1

Called by: Command Processor

Function: Errors that occur during the execution of the other Layout modules (Selecter, Connecter, Placer,Router) may require handling help from the Command Processor. If so, the error condition is communicated through the **error** parameter in the **ARGUMENT** file. The appropriate corrective action is initiated, and the **ARGUMENT** file is updated.

Entry Conditions: **error** indicates the cause of the error.

Exit Conditions: Action appropriate to the **error** will have been completed, and the **ARGUMENT** file will reflect the changes, if any.

Pseudo Code:

```
Begin Resolve Error
   Case error of:
       Error code - Error routine
           . . .
           . . .
   If error is external
      Then Reset error in ARGUMENT file
End.
```

Figure IV-3.3 Error Handler

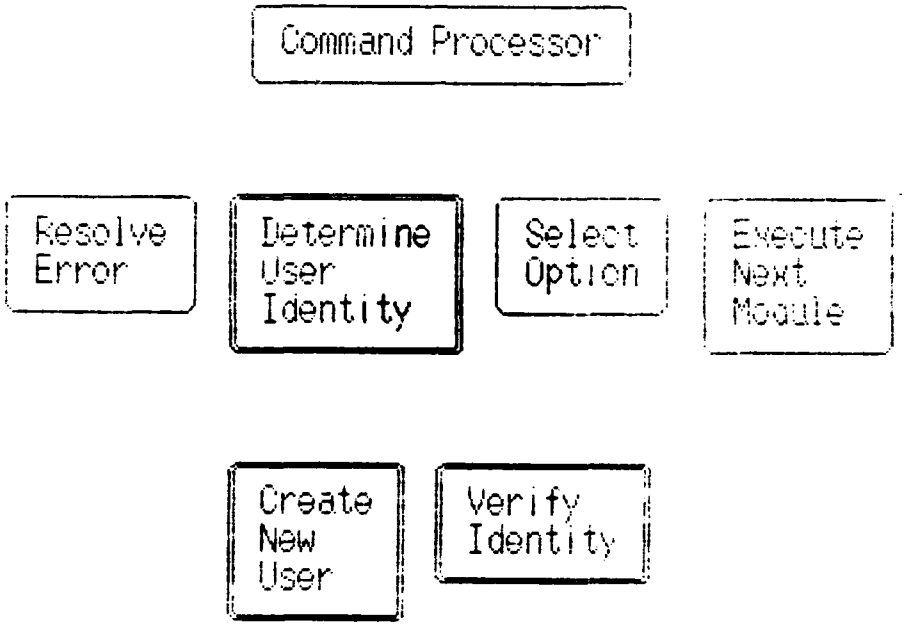


Figure IV-3.4 Determine User Identity Modules

Module Name: Determine User Identity Level: 1

Called by: Command Processor

Calls: Create New User, Verify Identity

Function: Asks the user to identify himself. The **name** is then looked up in the **LIST OF USERS**. Unknown users will be asked if they wish to be added to the system.

Entry Conditions: Command Processor has been entered for the first time and needs to know who the user is. No arguments are passed.

Exit Conditions: **valid user** is set if the user is valid. **current user** will identify who the user is.

Pseudo Code:

```
Begin Determine User Identity
   Repeat
      name = Query"Who are You?"
      Look up name in LIST OF USERS
      If name is not found
         Then typo = Query"Did you enter
                              your name correctly?"
   Until no typo
   If name is not found
      Then Create New User
      Else Verify Identity
End.
```

Figure IV-3.5 Determine User Identity

Module Name: Create New User Level: 2

Called by: Determine User Identity

Function: A new **user ID** and **password** will be assigned to the **name** and entered into the **LIST OF USERS.**

Entry Conditions: **name** is to be added to the system.

Exit Conditions: **valid user** will be set if the operator really did get added to the system. **current user** will contain the **user ID** assigned to **name.**

Pseudo Code:

```
Begin Create New User
   add = Query"Do you wish to be added as a new user?"
   If add is set
      Then Assign new user ID to name
           protect = Query"Do you want a password?"
           If protect is set
              Then password = Query" What will your
                                    password be?"
              Else password = Null
           Add user ID, name, password to LIST OF USERS
           Set valid user
           current user = user ID
      Else Reset valid user
           current user = Null
End.
```

Figure IV-3.6 Create a New User

Module Name: Verify Identity Level: 2

Called by: Determine User Identity

Function: A user has entered a **name** that is present in the **LIST OF USERS.** If he knows the password, then he will be allowed acces to his projects.

Entry Conditions: **name** of unvalidated user must be validated

---

Exit Conditions: **valid user** is set if the operator can prove who he is by typing the correct password. If he doesn't know it, he will not be allowed to do anything else.

Pseudo Code:

```
Begin Verify User
   Look up password of name in LIST OF USERS
   If password = Null
      Then Set valid user
           current user = user ID
      Else proof = Query"Prove it!"
           If proof = password
              Then Set valid user
                   current user = user ID
              Else Reset valid user
                   current user = Null
End.
```

Figure IV-3.7 Verify User Identity

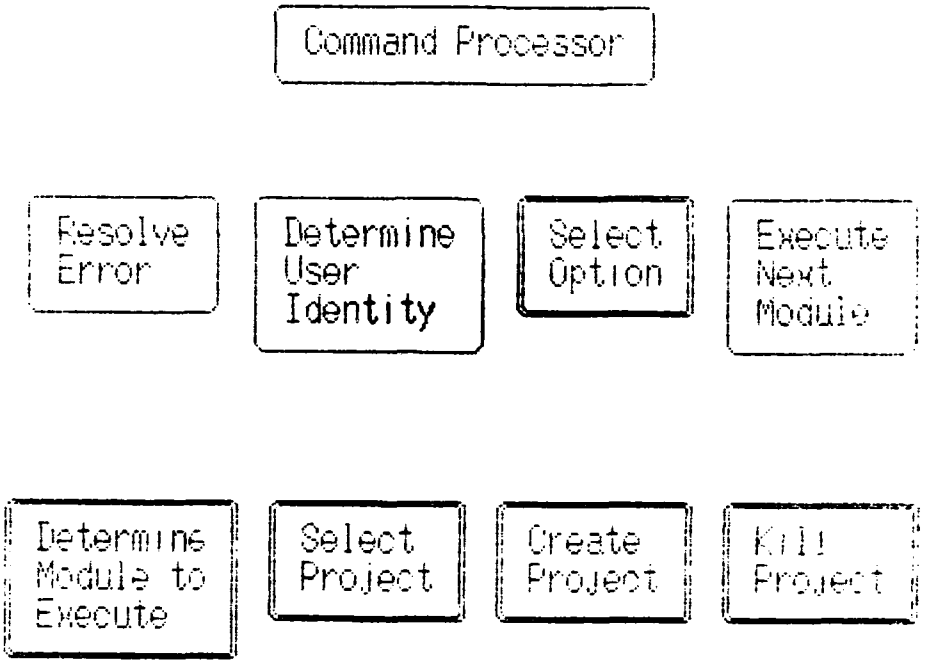


Figure IV-3.8 Select Option Modules

Module Name: Select Option Level: 1

Called by: Command Processor

Calls: Determine Module to Execute, Select Project, Create Project, Kill Project

---

Function: The user may either continue to work on the same project from start to finish, or may complete various projects one step at a time. After each major module is completed, the **current project** can be put aside and a new project can be selected, or the same project can be completed still further, or the user can quit for now and exit the program.

Entry Conditions: **valid user** is set and **current user** contains the **user ID** of the current user. The **ARGUMENT** file contains information about the current state of the **current project.**

Exit Conditions: The **ARGUMENT** file will have been updated to reflect the new (if any) **current project. next** module will indicate which module is to be executed next.

```
Pseudo Code:
Begin Select Option
   Update LIST OF PROJECTS from ARGUMENT file
   next module = Null
   Repeat
      Display Menu:
    * Continue with current project
    * Switch to another project
    * Create a new project
    * Discard current project
    * Exit to operating system
      Get choice
      Case choice of:
         Continue - Determine Module to Execute
         Switch   - Select Project
         Create   - Create Project
         Discard  - Kill Project
         Exit     - next module = Operating System
                    current project = Null
   Until next module isn't Null
   Update ARGUMENT file from LIST OF PROJECTS
End.
```

Figure IV-3.9 Main Menu Options

Module Name: Determine Module to Execute Level: 2

Called by: Select Option

Function: The **state of completion** of the **current project** indicates what has been completed so far. Previously completed steps may be retried with different parameters, or the next step may be selected. In any case, no module may be executed until the **state of completion** indicates that all required previous steps have been completed.

Entry Conditions: **current project** indicates which project is under consideration. The **LIST OF PROJECTS** contains the **state of completion.**

Exit Conditions: **next module** will identify which module is to be executed next for this project.

Pseudo Code:

```
Begin Determine Module to Execute
   Look up state of completion of current project
           in LIST OF PROJECTS
   Case state of completion of:
      Select Board          - highest module = Selecter
      Select Components     - highest module = Selecter
      Selection Complete    - highest module = Connecter
      Specify Components    - highest module = Connecter
      Connections Complete  - highest module = Placer
      Place Components      - highest module = Placer
      Placement Complete    - highest module = Router
      Route Connections     - highest module = Router
      Routing Complete      - highest module = Router
   Display Menu from Selecter to highest module
  * Selecter
  * Connecter
  * Placer
  * Router
   Repeat
      Get choice
      If choice < highest module
         Then proceed = Query "You sure you want
                                to Redo this?"
   Until proceed = OK
   next module = choice
End.
```

Figure IV-3.10 Module Selection Menu

Module Name: Select Project Level: 2

Called by: Select Option

Function: The user is asked to choose from among his current projects. **current project** may also be left as is, if a mistake was made. If no projects are assigned to the **current user**, then **current project** is set to Null.

Entry Conditions: **current user** identifies whose projects to look for in **LIST OF PROJECTS.**

Exit Conditions: **current project** identifies the new current project.

Pseudo Code:

```
Begin Select Project
   Look for current user = user ID in LIST OF PROJECTS
   For each match:
      Display Menu item project name
   If no match
      Then current project = Null
      Else Get choice
           switch = Query"Confirm switch from"
                              current project "to" choice
           If switch is set
              Then current project = choice
End.
```

Figure IV-3.11 Project Selection

Module Name: Create New Project Level: 2

Called by: Select Option

Function: A new entry in **LIST OF PROJECTS** will be created. The user will be asked to name the project and give a short description to aid in future identification. An entry in the **GLOBAL DIRECTORY** will be allocated for the **PROJECT** file. The new project will become the **current project.**

Entry Conditions: **current user** is to be assigned a new project.

Exit Conditions: Newly created project will be the **current project.**

Pseudo Code:

```
Begin Create New Project
   project name = Query"What do you want to call it?"
   project description = Query"Briefly describe it."
   Assign new project ID to project
   state of completion = Select Board
   user ID = current user
   Add project ID, user ID, project name,
       project description, state of completion to LIST OF
PROJECTS
    Allocate disk space for PROJECT file in GLOBAL
DIRECTORY
End.
```

Figure IV-3.12 New Project Creation

Module Name: Kill Project Level: 2

Called by: Select Option

Function: The user wishes to delete the **current project** from the **LIST OF PROJECTS.** After confirming this, the **PROJECT** file for the **current project**is deleted from the **GLOBAL DIRECTORY** and the disk space is de-allocated

Entry Conditions: **current project** is to be deleted

Exit Conditions: If confirmed, **LIST OF PROJECTS** and **GLOBAL DIRECTORY** will no longer contain information about the old project, and **current project** will be set to Null.

Pseudo Code:

```
Begin Kill Project
   If current project <> Null
      Then proceed = Query"Confirm deletion of"
                              current project
           If proceed is set
              Then Remove current project from
                           LIST OF PROJECTS
                   Remove PROJECT file of current project
                           from GLOBAL DIRECTORY
                   De-allocate disk space
                   current project = Null
End.
```

Figure IV-3.13 Project Deletion

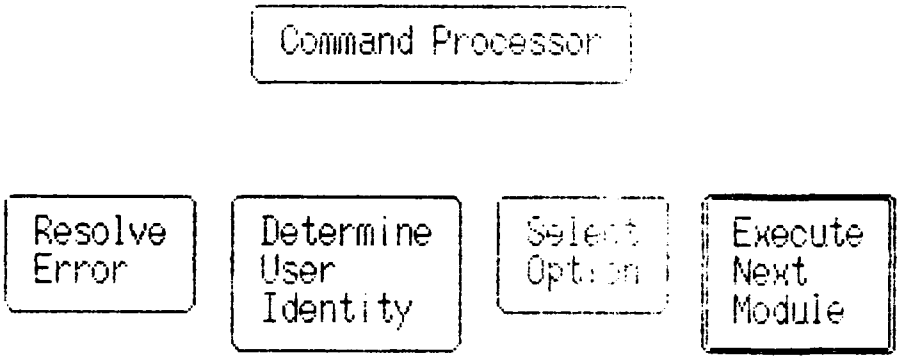


Figure IV-3.14 Execute Next Module

Module Name: Execute Next Module Level: 1

Called by: Command Processor

Executes: Selecter, Connecter, Placer, Router, Exits to Operating System

---

Function: The Layout modules cannot be simply called like a normal procedure, because they are not resident in memory at the same time with the Command Processor. In operation, the Command Processor chains to the **next module** which in turn chains back to the Command processor. Communication is through disk files. Before execution is attempted, the files required by the **next module** are looked up in the **GLOBAL DIRECTORY**. Their locations are loaded into the **ARGUMENT** file, and control is transferred to the **next module.**

Entry Conditions: **next module** identifies which module is to be executed. **GLOBAL DIRECTORY** contains information about which files the modules need to execute properly.

Exit Conditions: This is a "dead-end" routine because it does not return control to the caller. It is the responsibility of the module that is executed to return control to the Command Processor with the **ARGUMENT** file updated.

Pseudo Code:

```
Begin Execute Next Module
   Look up required files for next module
           in GLOBAL DIRECTORY
   For each required file:
      Add required file location to ARGUMENT file
   Look up resident filename of next module
           in GLOBAL DIRECTORY
   Execute resident filename
End.
```

Figure IV-3.15 Module Chaining

Graphics Routines

The following graphics routines have been implemented in very basic form for testing purposes, and the design is presented to satisfy the requirements for graphics capability as indicated in the general design of the layout modules in chapter III.

Basic Features

The basic graphics display routines are those required for pixel manipulation and changing of the display window. They will be very machine dependent, so the design of each depends on the target machine. Here is a description of the required routines and associated definitions:

pdux,pduy : Physical Display Units x and y represent the actual screen coordinates for the target machine.

plotcode : on = pixel will be turned on
off = pixel will be turned off
flip = pixel will be inverted
same = state of pixel will remain unchanged

quadrant : NW, NE, SW, SE each represent one quarter of the logical display space. To allow boards larger than the physical screen to be displayed, the logical display space is divided into four quadrants, only one of which may be displayed at once.

Pixel Manipulation Routines

set scale (scale) : The system scale factor is a real variable and may be set to any value desired. Once set, any subsequent coordinates will be scaled accordingly. The initial value is one.

set coord (x, y) : A coordinate in terms of mils is converted into the corresponding physical display coordinates using the current value of the system scale

factor.

pixel (plotcode) : The pixel at the current location (determined by set coord) will be modified according to the value of plotcode.

pixelset : A boolean function that returns the state of the pixel.

limits (quadrant, xmin, xmax, ymin, ymax): A procedure that returns the minimum and maximum x and y values (in mils) that will be visible in the given quadrant.

Window Manipulation routines

window (quadrant): The quarter of the virtual display screen specified by quadrant will be physically displayed. Depending on the display hardware, the current screen may be swapped out to disk.

text : The graphics screen will be replaced with the text screen. The graphics will not be erased, but may be swapped out to disk if necessary.

loadgraphics : The graphics display memory will be loaded from the graphics dump file.

savegraphics : The graphics display memory will be saved into the graphics dump file.

cleargraphics : The graphics display memory will be cleared completely.

Advanced Features

The following procedures will assume the basic features are defined and available. They include line and box drawing routines, copy routines, and cursor movement control

routines. They will be implemented using standard Pascal and the basic graphics routines, but if the display hardware supports some of these features, then these routines may be re-written to take advantage of the extra capabilities.

The following definitions apply to these functions:

point	: screen location in mils. The x and y coordinates can be referred to individually as point.x and point.y. Note that the graphics display logical screen is four times larger than the physical screen, so the point may not necessarily be on the current physical display.
layer	: 1..4 indicating which layer is being drawn on.
overlay	: true = the existing points which are set will be left alone when new points are added. false = any existing points will be reset before the new points are added.
orientation	: asis = don't change relative orientation clockwise = rotated 90 degrees right counter = rotated 90 degrees left upsidedown = flipped upside down

Drawing commands

line (point1, point2, plotcode, layer): A line will be drawn on the display between points 1 and 2. All of the pixels along the line will be affected according to the value of plotcode. Each layer will be drawn with different line styles (solid, dotted, dashed, etc.) The current quadrant will be switched if necessary.

drawbox (point1, point2, plotcode): A box will be drawn using points 1 and 2 as opposite corners. The box will always be drawn using solid lines.

copy (point1, point2, point3, overlay, orientation): The contents of the box defined by points 1 and 2 will be copied to the box of the same size with point 3 as the lower left corner. If overlay is true, then the current contents of the destination box will not be erased before the source box is added. Orientation will define how the destination box will be oriented with respect to the source box.

move (point1, point2, point3, overlay, orientation): Same as copy except that the source box will be reset.

draw (symbol, point, orientation): The symbol definition will be looked up in the SYMBOL FILE and will be transferred to the display with point as the lower left corner. The orientation will be changed accordingly.

size (symbol, dx, dy): The size of the symbol will be returned in dx and dy.

defsymbol (symbol, point1, point2): The box defined by points 1 and 2 will be added to the SYMBOL FILE with the label symbol.

Cursor Motion

getcoor (point1, point2, point3): A cursor will appear on the screen at the center of the box defined by points 1 and 2. The user will be able to move the cursor around the screen within the box until he presses RETURN. The current cursor location will then be returned as point 3. The cursor will be non-destructive and the user will be able to define the resolution of cursor motion dynamically.

Chapter V

Implementation

Two different micro-computers were used to develop the software for this project. They use different operating systems and different Pascal compilers, providing a good portability test.

One of the computers is an LNW-80, which is compatible with the Radio Shack TRS-80 Model I computer. It has 48K bytes of RAM and four floppy disk drives with a total storage space of over 1 Megabyte. The graphics display is 480 pixels horizontally and 192 pixels vertically within a display space of 7.5 by 4.5 inches, which gives a worst case resolution of 24 mils. This is adequate to represent a 50 mil grid in actual size. The operating system is Dosplus 3.5, unique to TRS-80 computers and their equivalents.

The other computer is an Epson QX-10. Although equipped with 256K bytes of memory, only 64K is accessable when using the CP/M operating system. The graphics display is 640 X 400 pixels within an 8" X 5" screen, yielding a resolution of 12.5 mils.

Simultaneous software development proved to be very time-consuming because of the iterative nature. Improvements and additions made to one version had to be made to the other version as well, which made keeping track of source code files a non-trivial task.

The present implementation of the layout system is not as complete as originally intended, due to unanticipated problems with the Pascal file handling system and with manipulation of the QX-10 graphics hardware. The available documentation is not complete, and the other problems slipped the schedule enough to make experimentation impractical.

The next section explains the differences between the two compilers used and will allow comparison of the source code.

Pascal MT+ on Epson QX-10

Random Access Files

Standard Pascal only defines sequential access files, and MT+ extends this definition to allow for random access through the use of two procedures - seekread and seekwrite. Standard Pascal procedures are used to open the file, however. This leads to problems when all file accesses are to be handled through a common set of procedures, because the file record structure must be known at the time the file is opened.

A generalized file access routine should not have to know the actual record structure of the file, just how many bytes to transfer. The solution was to use a fixed-size record for all file access, declared as an array of characters. Procedures were then written to perform record blocking and de-blocking within this maximum record. This requires the file routines to maintain the actual record

length for each file, so only the appropriate number of bytes are transferred.

External Procedures and Variables

The two compilers have almost identical facilities for handling external declarations, but the syntax is totally different. As a result, none of the routines may be compiled "as is", but the changes between them are mechanical in nature.

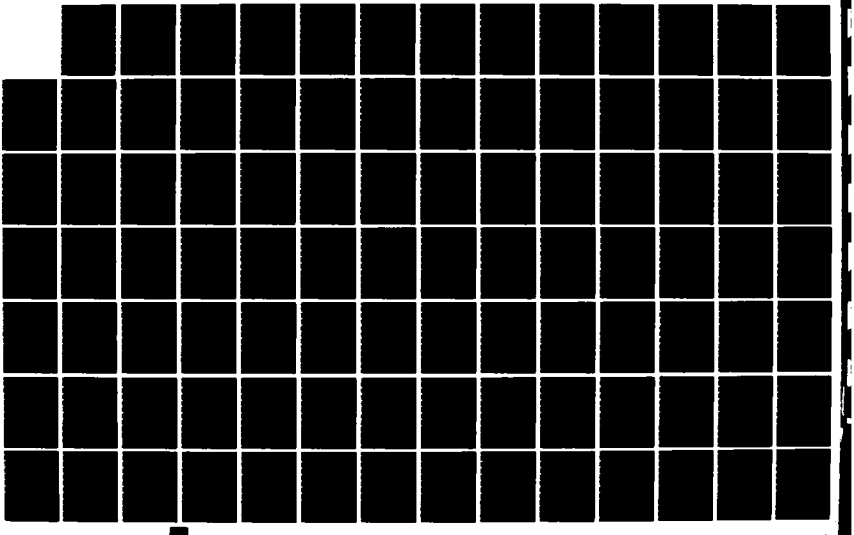
MT+ defines two keywords, MODULE and MODEND. to differentiate between a program module and a set of procedures being compiled separately. To allow procedures and functions defined in one module to be accessed from another, the procedure or function heading is preceded by the keyword EXTERNAL if the definition exists in another module or program. Variables are handled similarly, with the EXTERNAL keyword preceding the type of the variable.

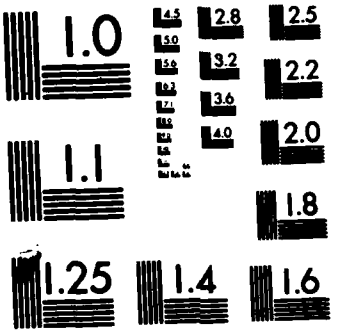
Static vs. Dynamic Variables

In standard Pascal, variables do not exist unless the procedure in which they are defined is currently active. In separately compiled modules, MT+ defines all variables declared outside the procedures to be static, and they are treated just like global variables defined in the main program. This allows modules to declare variables that will not become undefined when the procedure using them finishes executing.

AD-A138 427 PRINTED CIRCUIT BOARD LAYOUT BY MICROCOMPUTER(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING E W KRAUSMAN DEC 83 AFIT/GE/EE/83D-35 2/3

UNCLASSIFIED F/G 9/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

An MT+ Module with Static variables has the following format:

```
MODULE module name;
VAR static variable name : type;
    variable declared elsewhere : EXTERNAL type;

{ References to procedures defined elsewhere follow }
EXTERNAL procedure name(parameters);

{ Procedures being compiled follow }
PROCEDURE procedure name(parameters;
VAR { all variables needed by the procedure }
BEGIN
{ Procedure body }
END;
...
...
MODEND.
```

LNW (TRS-80 Model I compatable) and Alcor Pascal

Random Access Files

Unlike MT+, Alcor defines separate procedures for opening and closing random access files, as well as reading and writing them. The record length of the file is passed as a parameter to the open subroutine, and the address of a variable is passed to the random read and write routines. This means that the file routines do not need to know the structure of the file, and any record length can be used with ease.

Separate Compilation

Alcor uses a compiler option to tell the compiler not to generate code for the body of the program, called the {$NULLBODY} option. Basically, the entire program consists of the statements BEGIN {$NULLBODY} END. instead of MODEND. as with MT+.

External Procedure Declarations

The only difference here is the position of the keyword EXTERNAL. Alcor Pascal places it after the procedure or function heading, instead of before it.

Static Variables

Two new keywords, COMMON and ACCESS are used to do the same thing that external variable declarations do in MT+. COMMON is used instead of VAR to indicate that the varaibles being defined are to be allocated statically, and the keyword EXTERNAL is not used. In every procedure that uses a COMMON variable, the variable must appear in an ACCESS statement.

The format of an Alcor module is as follows:

```
PROGRAM program name;

{ The COMMON statement is the same regardless of where
the variable is defined.}
COMMON static variable name : type;

{ Procedures defined elsewhere go here }
procedure name(parameters); EXTERNAL;
...
...
{ procedure definitions being separately compiled go
here }
PROCEDURE procedure name(parameters);
ACCESS variable names declared as common;
VAR local variables : types;
BEGIN
{ Body of procedure }
END;
...
...
BEGIN
{$NULLBODY}
END.
```

Appendix A explains the basic linked-list file format, and Appendix B describes the file formats used by the Command Processor Shell and the ARGUMENT FILE. To make installation of the Command Processor easier, Appendix C describes the Installation utilities that have been written.

The complete source code for the Command Processor Shell is presented in two versions, one for each computer-compiler pair, in Appendixes D and E.

Appendix F contains a users manual for the Command Processor which explains the use of the Query, Menu, and Help functions.

Chapter VI

Results

File Size Limitations

Because all of the information is stored in files, project complexity will be limited by the amount of disk space available. All of the files required by a particular module should fit on line at the same time, to minimize the amount of disk swapping required. The amount of data that can be maintained for a project depends on the following factors - the physical storage limitation of the disk itself, and the amount of overhead required by the system.

File overhead refers to the amount of file space that isn't available for storing actual data. For linked files, the overhead imposed is four bytes per record, independent of the record length. Longer records will have less overhead on a percentage basis than short records. For example, an eight byte record will consist of four bytes of pointers and four bytes of data, with a resultant overhead figure of 1/2. A 128 byte record will have the same four pointer bytes with 124 bytes of data. The overhead in this case, 1/32, is much lower.

Most of the layout module record lengths will probably lie between these two extremes. A 16 byte record could be used to store circuit trace information, of which 12 bytes would be available for data. Allowing for 2,000 traces per layer on a four layer board would require 8,000 16 byte records, or 128,000 bytes.

As long as all of the records in a file contain the same type of information, calculation of file overhead is simple. Problems arise when variable length records are stored in one file, because the longest record length is used for each record. The unused bytes at the end of each shorter record must be added to the total overhead. Any bytes used to distinguish between the types of records in the file also add to the overhead. The total overhead figure will include the fixed four bytes per record, plus an amount depending on the mix of each of the logical record lengths.

Improvements to Existing Code

The current Command Processor shell does work correctly, but there is a feature that should be improved to provide a better user environment. When the user is asked a question, there is no provision for an escape - A valid answer must be provided before control returns to the caller. On-line help is available, but if the user is totally confused, or realizes that he answered the last question wrong, there is no way the indicate to the calling program an exception condition.

Proposed solution - the query modules should include an additional parameter to indicate that the user didn't desire to answer the question. This would allow the calling program to detect the exception condition and provide a way for the user to gracefully "back up" through a series of questions.

Future Developments

Because of the unforseen level of difficulty in writing truly portable Pascal programs, it was probably not the best choice for a project of this type. C appears to be a much more suitable language and conversion from Pascal to C should not be difficult for an experienced C programmer. When this project was started, I was unfamiliar with C and thought that it would take too much time to obtain and learn it.

The basic design is sound and I will be continuing to work on this project even though the formal thesis work may be completed. It will take quite a lot of work to fully implement the system, but the results should be worthwhile.

Bibliography

1. Calafiore and Foster, A System for Multilayer Printed Wiring Layout,Proc. 11th Design Automation Conference, 1974.

2. Brinsfield and Tarrant, Computer Aids for Multilayer Printed Wiring Board Design, Proc. 11th Design Automation Conference, 1974.

3. Patterson and Phillips, A Proven Operational CAD System for P.W.B. Design, Proc. 12th Design Automation Conference, 1975.

4. Nishioka, Kurimoto, and Nishida, A Minicomputerized Automatic Layout System for Two-Layer Printed Wiring Boards, Proc. 13th Design Automation Conference, 1976.

5. Pedro and Garcia, DOCIL: An Automatic System for Printed Circuit Board (PCB) Designing, Proc. 14th Design Automation Conference, 1977.

6. Matthews, A. J., A Human Engineered PCB Design System, Proc. 14th Design Automation Conference, 1977.

7. Bayegen, H. M., An Integrated System for Interactive Editing of Schematics, Logic Simulation and PCB Layout Design, Proc. 15th Design Automation Conference, 1978.

8. Stevens, vanCleemput, Bennett, and Hupp, Implementation of an Interactive Printed Circuit Design System, Proc 15th Design Automation Conference, 1978.

9. Villers, P., A Minicomputer Based Interactive Graphics System as used for Electronic Design and Automation, Proc. 15tn Design Automation Conference, 1978.

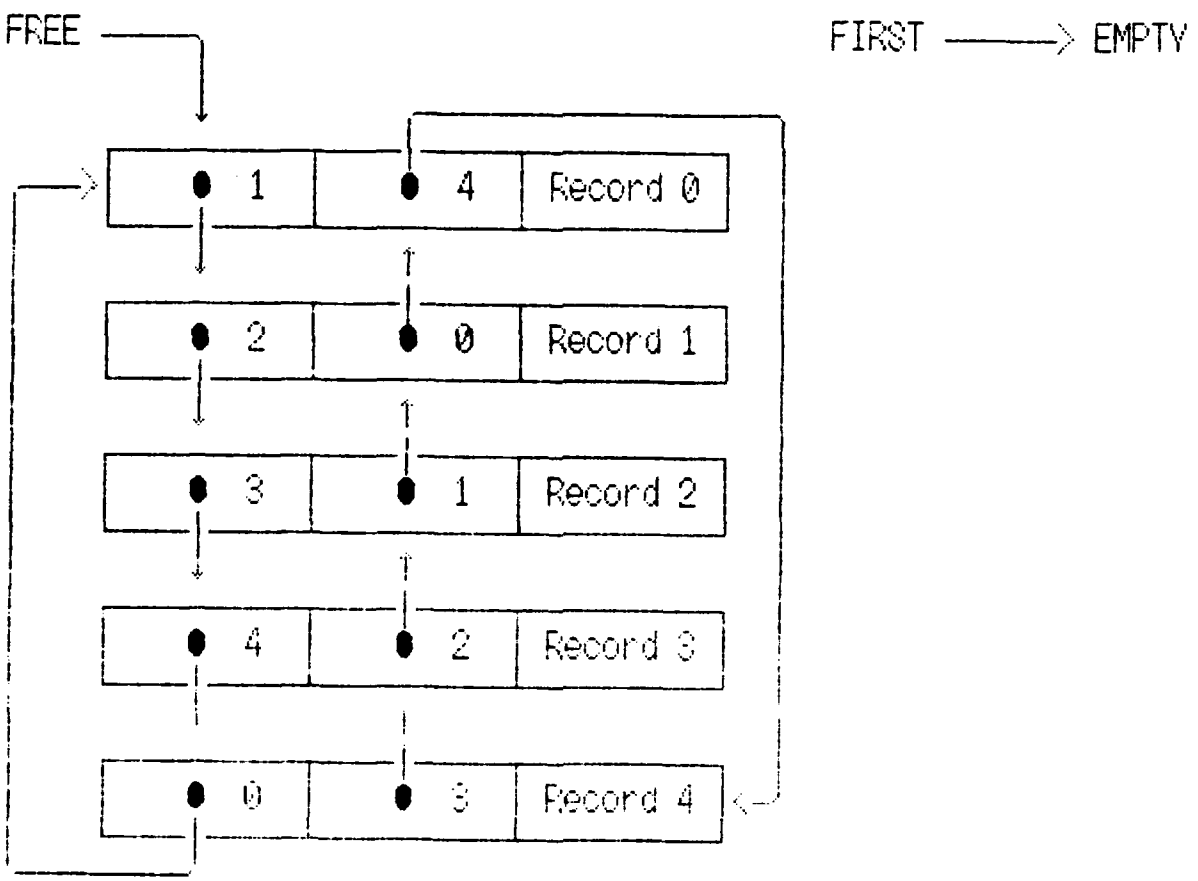
10. Johnson, D. R., PC Board Layout Techniques, Proc. 16th Design Automation Conference, 1979.

11. Shiraishi, Ishii, Kurita, and Nagamine, ICAD/PCB: Integrated Computer Aided Design System for Printed Circuit Boards, Proc. 19th Design Automation Conference, 1982.

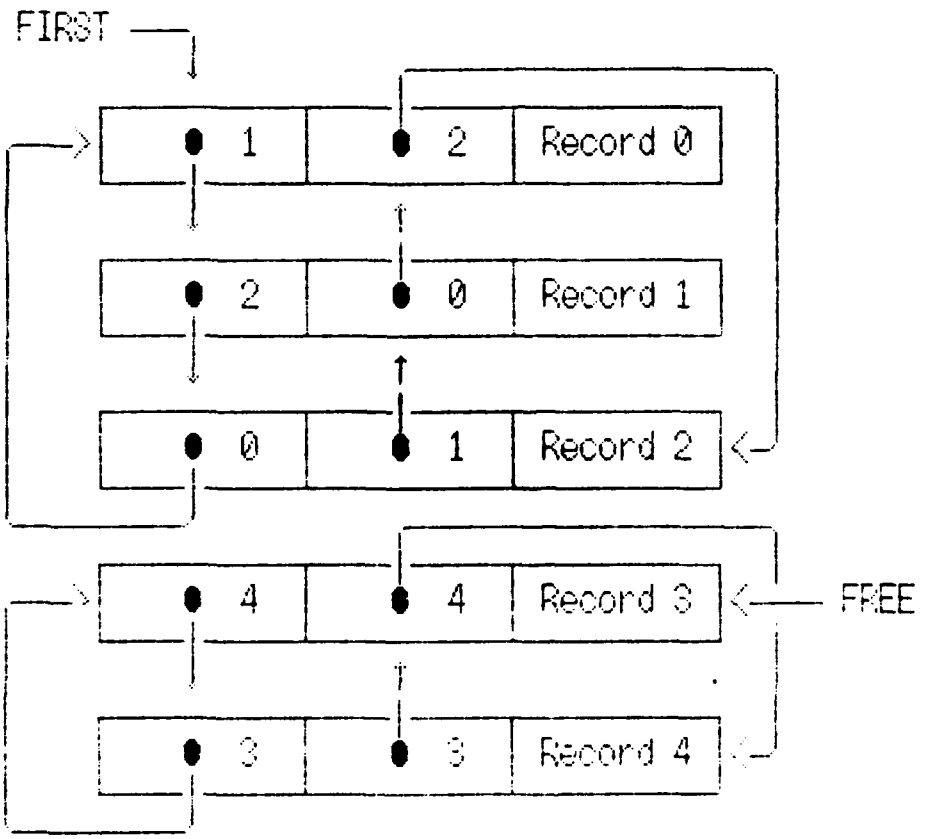
APPENDIX A - Linked File Format

All of the files maintained by the system have the same basic format - a linked list of fixed length data records, accompanied by another list of free records. When a file is created, all of the records are linked together and assigned to the Free list. Later insertions and deletions involve transfering a record from one list to another. No actual movement of the records in the file takes place.

For example, consider a file with five records allocated. After initial creation, it would look like this:



Notice that the free list is circular - the previous record pointer of the first record points to the last record, and the next record pointer of the last record points to the first record. The two pointers First and Free point to the first record of their respective lists, and a value of -1 for either pointer means there are no records in the list. In the present example, First is pointing to an empty list, because no records have been inserted yet. Here is how the file would look after three insertions:



Because the lists are circular, detecting an End of File condition becomes a little tricky. After a file has been Reset, two pointers, Next and Last, are both set to point to the record pointed to by First. The first Read will look at the first record and set Next to point to the next record. After four more Reads, Next will again be pointing to First, but Last will not. This is how an End of File is detected - whenever Next equals First and Next is different from Last, then we've wrapped around back to the top of the list. When there is only one record in the list, however, this scheme doesn't work. A one record list always points to itself, because it is both the next record and the last record as well as the first record. In this case, an End of File will never be indicated. The solution is to avoid single element lists.

Slots

The LinkFile module maintains ten slots for file information. To access a file, a slot must be initialized for it. The initialization defines the filename, diskid, drive, and current pointer values for the file. Once initialized, the system automatically opens and closes files as necessary, and will prompt for disk switches when necessary.

When a module finishes execution, the current values of the pointers must be saved somewhere so that the next module may access them. The global directory maintains all variable file slot information for the command processor,

and the argument file is used to pass this information between modules. To ensure file integrity, the following procedure is followed: Initialize all necessary file slots by reading the information from a known source (either the global directory or the argument file). When done with the files, update the appropriate source to save the new state of the files.

The actual information maintained for each slot is system dependent, and can be found in the source code listings in appendices D and E.

APPENDIX B - Command Processor Files

The Command Processor uses six files to keep track of the users, projects, and disk allocation, along with menus, help, and communication with the other modules.

List of Users

Three pieces of information are retained for each user - a name, password, and a user id. Upon initial entry to the system, a match with one of the names already in the list is attempted. If the match is successful, then the user must also match the password, unless the password is all blanks. Here is the format of the Users file:

```
Name     : String containing UPPER CASE name of user
ID       : positive integer greater than zero
Password : String containing password defined by user
```

List of Projects

Project information includes the project name, an additional description, a progress indication, and the owner of the project, all identified by a unique project ID.

```
ID         : positive integer greater than zero
Name       : String containing user defined name for
             project
Desc       : Optional description of project
Completion : Single byte integer representing the level
             of completion
User       : Single byte integer identifying the owner
             of the project
```

List of Disks

As projects are created, diskette space is reserved in advance for each project file. The list of disks keeps track of current space available on each disk. Each project is assigned a unique disk or disks, so no disk will contain files from more than one project. When projects are

deleted, the disk space is reclaimed for the owner of the project. This ensures that physical disks always belong to one owner, allowing each user to keep his own disks separately.

Disk ID	: An integer identifying the disk
Assigned	: A flag indicating whether or not the disk currently belong to a project, or is available for re-assignment.
User or Project ID	: Depending on the Assigned flag, this field identifies either the owner of an un-assigned disk, or the project to which the disk has been assigned.

Menus

The file of menus has a structure which allows a menu to be represented as a linear list regardless of the number of levels. All of the items belonging to one menu sequentially follow each other. As each item is read from the file, two flags, Bump Up and Bump Down, indicate whether the next item is at a lower level, higher level, or the same level. The end of a menu is indicated when the level reaches zero.

Menu Number	: Indicates which menu this record belongs to
Bump Down	: A flag indicating that the next record is at a lower level
Bump Up	: A flag indicating that the next record is at a higher level
Item Text	: The text displayed to the user on the screen
Item Code	: Code number for this selection
Help Index	: Which help message to display for this selection

Help File

The help file is very simple in structure - each record contains two fields: an index number, and a line of a help message. When help is asked for, the file is scanned until the first matching index number is found, then all sequential records are displayed until a different index number is found.

Help Index : Integer indicating to which message this record belongs
Line : One line of the help message

Argument File

The argument file contains a header record with a different format than the rest of the entries. The header information keeps track of the current state of the system, and the rest of the records contain information about each file which is passed between the modules.

Here is the format of the header record:

Next ID	: The next disk ID to assign
Next User	: The next available user ID
Next Project	: The next available project ID
Project ID	: The ID of the current project
User ID	: The ID of the current user
Error Code	: Current error condition, if any
Completion	: Number representing the state of the current project
Module	: Number representing the last module executed

The file entry format is a subset of a global directory entry:

File Num	: Slot number occupied by the file
File Name	: System dependent file name
Linked	: Flag indicating whether or not the system will maintain the file as a linked list
Drive	: Drive on which to mount disk containing the file
Disk ID	: Identifier of disk containing the file
Recs Avail	: The amount of free space left in the file
Rec Len	: Logical record length of the file
First	: Pointer to the first logical record
Free	: Pointer to the free list

APPENDIX C - Installation

In order for the Command Processor to work, four files must be initialized - the global directory, the menu file, the help file, and the argument file header. Because these files use Pascal structures, programs were written to read in conventional ASCII files and convert them to the format required by the Command Processor. Described below is the format of the ASCII files required by each of the programs.

GDIRGEN - Generate the Global Directory

GDIRGEN reads a file called GDIRGEN.DAT and creates a new directory entry for each line in the file. Each line has fourteen fields with the following format:

Field Name	Field Length	Contents
CP	1	T or F
Selecter	1	T or F
Connecter	1	T or F
Placer	1	T or F
Router	1	T or F
OS	1	T or F
Separator	1	Anything
Linked	1	T or F
Separator	1	Anything
File Name	12	System File Name
Separator	1	Anything
Drive	1	Drive Identifier
Sepatator	1	Anything
Slot Number	variable	1 to 10
Project ID	variable	0 or -1
How Many	variable	Number of Records
Disk ID	variable	0 or 1
Record Length	variable	1 to 128

The first six fields are flags indicating if the file will be accessed by the corresponding module, and the variable length fields are all numeric values separated by spaces.

Project ID and Disk ID are set to -1 and 0, respectively, to create a template entry. Template entries do not correspond to actual files, but are used when new project files are created. The global directory is searched for these template entries, and a corresponding real file and entry is generated, with the Project ID and Disk ID fields set to actual values.

MENUGEN - Generate the Menu File

The contents of MENUGEN.DAT are used to initialize the system menu file. Each line corresponds to a single menu item or a new menu number. The first cnaracter of a menu item is used to control the nesting level - a V will go down one level, and a ^ will go up one level. When the level reaches zero, the menu is complete, and the next line should contain another menu number or a zero to indicate the end of the file.

Field Name	Field Length	Contents
Level	1	^ or V or space
Separator	1	Anything
Item Text	20	Displayed Text
Separator	1	Anything
Item Code	variable	0 to 255
Help Index	variable	Help Message Number

HELPGEN - Generate the Help File

The contents of HELPGEN.DAT are used to create the system help file. Each line corresponds to one line of a help message. Help messages may be an arbitrary number of lines long.

Field Name	Field Length	Contents
Help Line	30	A line of the message
Separator	1	Anything
Help Index	variable	Help Message Number

ARGSGEN - Initialize the Argument File

The Command Processor reads the Argument file every time is entered to determine the current state of the system. This program initializes the Argument File header so the Command Processor won't bomb out the very first time it is executed. No input data file is used.

APPENDIX D - QX10 Source Code

```
{*****************************************************************}
{* TermIO Module - Terminal I/O support routines                 *}
{*****************************************************************}
{* All of the routines use one byte control codes for the        *}
{* various screen operations and expect one byte responses       *}
{* from the keyboard.  These control bytes must be translated    *}
{* into the actual control sequences for the terminal being      *}
{* used.                                                         *}
{*****************************************************************}
module termio;
const q_x           = 10; { Define Screen Areas }
      q_y           = 20; {Query Box }
      q_lines       = 3;
      m_x           = 5;  {Menu Box  }
      m_y           = 1;
      m_lines       = 20;
      h_x           = 40; {Help Box  }
      h_y           = 1;
      h_lines       = 20;
      max_prompt    = 30; { Maximum Length of Character string }

{ Define codes used internally }
      cursorleft    = $18;
      cursorright   = $19;
      cursorup      = $1B;
      cursordown    = $1A;
      clear_to_eol  = $1E;
      leftarrow     = $08;
      rightarrow    = $09;
      uparrow       = $5B;
      downarrow     = $0A;

type

      boxes         = (q, m, h);
      char_string   = array[1..max_prompt] of char;

{ Use direct console I/O for CP/M systems }
{ @BDOS is a library function provided by the compiler }
external function @bdos(func:integer; parm:word):integer;

{*****************************************************************}
{* Procedure : WriteCH - Send a character to terminal            *}
{* Parameters : CH                                               *}
{* Entry Conditions : CH is either an ASCII character or one     *}
{* of the one byte control codes to be sent to the terminal.     *}
{* Process : If the character is one of the recognized control   *}
{* bytes, then it is converted to the string required by the     *}
{* particular terminal.  Otherwise, it is passed without         *}
{* translation directly to CP/M.                                 *}
{*****************************************************************}
```

TermIO Module

```
procedure writech(ch : char);
   var i : integer;
   begin
      case ord(ch) of   { Terminal Dependent Codes }
           cursorleft : i:=@bdos(6,wrd(8));
          cursorright : i:=@bdos(6,wrd(12));
             cursorup : i:=@bdos(6,wrd(11));
           cursordown : i:=@bdos(6,wrd(10));
         clear_to_eol : begin
                           i:=@bdos(6,wrd(27));
                           i:=@bdos(6,wrd(ord('T')));
                        end;
      else i:=@bdos(6,wrd(ord(ch)));
      end;
   end;

{*****************************************************************}
{* Function : GetKey - Get a character from the keyboard       *}
{* Result : ASCII character or one byte control code entered   *}
{* from the keyboard.                                          *}
{* Process : The keyboard is scanned until the operator        *}
{* presses a key.  If the terminal being used generates        *}
{* multiple codes for some keys, then this procedure must      *}
{* decode the sequence and return the appropriate one byte     *}
{* control code.  Many terminals, for example, use escape      *}
{* sequences for function keys and the arrow keys.  These      *}
{* strings must be converted into a single byte.               *}
{*****************************************************************}
function getkey : char;
   var i : integer;
   begin
      { Wait for a key to be pressed }
      repeat i:=@bdos(6,wrd($FF)) until i<>0;
      case i of
{ QX-10 }  11 : i:=uparrow;
{ arrow }   8 : i:=leftarrow;
{ keys  }  12 : i:=rightarrow;
           10 : i:=downarrow;
      end;
      getkey:=chr(i);
   end;

{*****************************************************************}
{* Procedure : GotoXY - Direct cursor positioning              *}
{* Parameters : X, Y                                           *}
{* Entry Conditions : X and Y determine where the cursor is    *}
{* to be positioned on the text screen.  The upper left        *}
{* is 0,0.                                                     *}
{* Process : The cursor will be moved to the position X, Y.    *}
{*****************************************************************}
procedure gotoxy(x,y : integer);
   var i : integer;
```

TermIO Module

```
   begin
    { QX-10 requires the following escape sequence :
      ESC,=,y+32,x+32 }
      i:=@bdos(6,wrd(27));
      i:=@bdos(6,wrd(ord('=')));
      i:=@bdos(6,wrd($20+y));
      i:=@bdos(6,wrd($20+x));
   end;

{*****************************************************************}
{* Procedure : WaitKey - Wait for a key to be pressed          *}
{* Process : This procedure is used to allow the operator to  *}
{* indicate he is ready to proceed with some operation.  No   *}
{* value is returned.                                         *}
{*****************************************************************}
procedure wait_key;
   var dummy : char;
   begin
      dummy:=getkey;
   end;

{*****************************************************************}
{* Procedure : Goto_Box - Move to specific screen area        *}
{* Parameters : Box, X,Y                                      *}
{* Entry Conditions : Box identifies which of the three screen*}
{* areas the cursor is to be positioned in.  X and Y are the  *}
{* coordinates relative to the upper left corner of the box.  *}
{*****************************************************************}
procedure goto_box(box : boxes; x,y : integer);
begin
   case box of
   q : gotoxy(q_x+x,q_y+y);
   m : gotoxy(m_x+x,m_y+y);
   h : gotoxy(h_x+x,h_y+y);
   end;
end;

{*****************************************************************}
{* Procedure : Clear_Line - Clear a line of a box             *}
{* Parameters : Box, Y                                        *}
{* Entry Conditions : Box identifies which of the screen areas*}
{* is to be affected.  Y is line line to be cleared.          *}
{*****************************************************************}
procedure clear_line(box : boxes; y : integer);
begin
   goto_box(box,0,y);
   writech(chr(clear_to_eol));
end;

{*****************************************************************}
{* Procedure : Clear_Box - Clear an entire screen area        *}
{* Parameters : Box                                           *}
```

TermIO Module

```
{* Entry Conditions : Box identifies which screen area is to  *}
{* be cleared.                                                *}
{************************************************************}
procedure clear_box(box : boxes);
var line, lines : integer;
begin
   case box of
   q : lines:=q_lines;
   m : lines:=m_lines;
   h : lines:=h_lines;
   end;
   for line:=0 to lines do clear_line(box,line);
end;

{************************************************************}
{* Function : Get_Count - Determine the size of a screen area *}
{* Parameters : Box                                           *}
{* Returns : The length of the screen area in lines.          *}
{* Entry Conditions : Box identifies the screen area to use.  *}
{************************************************************}
function get_count(box : boxes) : integer;
begin
   case box of
   q : get_count:=q_lines;
   m : get_count:=m_lines;
   h : get_count:=h_lines;
   end;
end;

{************************************************************}
{* Procedure : Input_Err - Signal an error condition to user  *}
{* Parameters : Err_Msg                                       *}
{* Entry Conditions : Err_Msg is the text of the message to   *}
{* displayed to the operator.                                 *}
{* Process : The Error Message is displayed below the menu    *}
{* box and the user is asked to acknowledge the message by    *}
{* pressing any key.  Control will not return until a key is  *}
{* pressed, after which the message will be erased.           *}
{************************************************************}
procedure input_err(err_msg : char_string);
begin
   clear_line(m,m_lines+1);
   write(err_msg);
   write(' Press any key.');
   wait_key;
   clear_line(m,m_lines+1);
end;

{************************************************************}
{* Procedure : Init_Term - One time terminal initialization.  *}
{* Process : Whatever initialization is required is done here.*}
{************************************************************}
```

TermIO Module

```
procedure init_term;
begin
   write(chr(26)); { Clear the screen }
end;

modend.
```

QueryYN Module

```
{*****************************************************************}
{* Yes-No Query Module - Accept yes or no answer from user       *}
{*****************************************************************}
{* A question is asked of the user which requires either a       *}
{* yes or no response.  The question text is supplied by the     *}
{* calling routine. The user responds by typing either a 'Y'     *}
{* or an 'N' (either upper or lower case) or a '?'.  All         *}
{* other characters are ignored.  The user types <RETURN>        *}
{* when his selection is complete.                               *}
{*****************************************************************}
module query_yes_no;
const
     cursoron = $0E;
    cursoroff = $0F;
       return = $0D;
       escape = $03; {cntl C}
   max_prompt = 30;      { Maximum length of a prompt }

type
{ Query Type Definitions }
     prompt_string = packed array[1..max_prompt] of char;
             p_len = 1..max_prompt;
             yesno = (yes, no, i_dunno);
             boxes = (q, m, h);

{ See the TermIO Module for details about the following }
external procedure clear_box(box : boxes);
external procedure goto_box(box : boxes; x,y : integer);
external procedure writech(ch : char);
external function getkey : char;

{ See the Help Module for details about the following }
external procedure help(help_index : integer);

{*****************************************************************}
{* Procedure : Query_YN - Get Yes or No response from user.      *}
{* Parameters : Prompt, Prompt_Length, Answer, Help_Index        *}
{* Entry Conditions : Prompt is a fixed length character         *}
{* string containing the text of the question to ba asked, and*}
{* Prompt_Length defines how many characters of Prompt to dis-*}
{* play to the user.  Help_Index is the help message             *}
{* associated with this question.                                *}
{* Exit Condition : Answer will contain eitner Yes or No if      *}
{* the user made a selection, or I_Dunno if the user asked       *}
{* help and didn t make a selection.                             *}
{*****************************************************************}
procedure query_yn(prompt : prompt_string;
            prompt_length : p_len;
               var answer : yesno;
               help_index : integer);

   var    character : char;
```

QueryYN Module

```
                    i : p_len;


   begin { qyery_yn }
      answer:=i_dunno;
      clear_box(q); writech(chr(cursoroff));
      goto_box(q,0,0);
      for i:=1 to prompt_length do
          writech(prompt[i]);
      goto_box(q,0,1);
      write('Please Respond with Yes or No.');
      repeat
         goto_box(q,prompt_length,0);
         character:=getkey;
         case character of
              'Y','y' : begin
                           answer:=yes;
                           write(' Yes          ')
                        end;
              'N','n' : begin
                           answer:=no;
                           write(' No           ')
                        end;
                  '?' : begin
                           answer:=i_dunno;
                           write(' I Don''t Know!');
                           help(help_index)
                        end;
         end;
      until (character=chr(return));
      writech(chr(cursoron));
      clear_box(q);
   end;

modend.
```

QueryNum Module

```
{*****************************************************************}
{* Number Query Module - Accepts number responses from the      *}
{*                       operator.                               *}
{*****************************************************************}
{* A Prompt supplied by the calling program is displayed to     *}
{* the user, along with the expected response value limits.     *}
{* All numeric characters typed by the user will be accepted    *}
{* as the response, terminated by a <RETURN>. All non-numeric   *}
{* characters will be ignored, and <RETURN> will be ignored     *}
{* if the value is out of range.                                *}
{* The following editing features are available to the user:    *}
{* Character Insert - typing control-S will cause all           *}
{*                    characters to the right of the cursor     *}
{*                    to move to the right.  Any character at   *}
{*                    the last position will be lost if the     *}
{*                    number is already 10 characters long.     *}
{* Character Delete - typing control-D will cause all           *}
{*                    characters to the right of the cursor     *}
{*                    to move to the left, deleting the char-   *}
{*                    acter under the cursor.                   *}
{* Sign Change      - At any time, typing a '-' will make the   *}
{*                    number negative, and a '+' will make it   *}
{*                    positive.  The default is positive.       *}
{*                    It doesn't matter where the cursor is     *}
{*                    when the sign is changed.                 *}
{* Cursor Motion    - The left and right arrow keys (or their   *}
{*                    equivalents) will cause the cursor to     *}
{*                    move in the appropriate direction.  The   *}
{*                    cursor location when <RETURN> is pressed  *}
{*                    is not important.  All visible charac-    *}
{*                    ters will be in the response.             *}
{* Other Features include specific help messages and units      *}
{* conversion.                                                  *}
{*****************************************************************}
module numeric_query;
const
    leftarrow = $08;  { Define arrow keys }
   rightarrow = $09;
   cursorleft = $18;  { Define cursor motion commands }
  cursorright = $19;
          ins = $13; {cntl S}   { Define Editing Commands }
          del = $04; {cntl D}
       change = $15; {cntl U}
       return = $0D;
         null = $00;
   max_prompt = 30;      { Maximum length of a prompt }
      max_num = 10;      { Maximum length of a number }

type
{ Query Type Definitions }
             unit = (mils, inches, mm, scalar);
    prompt_string = packed array[1..max_prompt] of char;
```

QueryNum Module

```
            p_len = 1..max_prompt;
            boxes = (q, m, h);

{ Screen I/O Declarations }
{ See the TermIO Module for details on the following }
external procedure goto_box(box : boxes; x,y : integer);
external procedure clear_line(box : boxes; y : integer);
external procedure clear_box(box : boxes);
external procedure writech(ch : char);
external procedure wait_key;
external function getkey : char;

{ See the Help Module for details on the following }
external procedure help(help_index : integer);

{*****************************************************************}
{* Procedure : QueryNUM - Accept Number from user                *}
{* Parameters : Prompt, Prompt_Length, MIN, MAX, Answer,         *}
{* Units, Help_Index                                             *}
{* Entry Conditions : Prompt is a fixed length character         *}
{* string which contains the question being asked.  Prompt_      *}
{* Length is the number of characters of Prompt to display.      *}
{* MIN and MAX describe the valid range of the number in mils.   *}
{* All internal numbers are stored in units of mils (or as       *}
{* unitless scalars).  Units represents the default units        *}
{* used for display only. Help_Index is the specific message     *}
{* associated with this query.                                   *}
{* Process : MIN and MAX are converted from mils to Units.       *}
{* and displayed beneath the Prompt.  The User's response is     *}
{* compared with MIN and MAX to verify it.  A valid response     *}
{* is converted from Units to mils and is returned to the        *}
{* caller. The user may change the default units at any time     *}
{* by pressing control-U until the desired units appear.         *}
{* Exit Conditions : Answer will contain a number between MIN    *}
{* and MAX mils (or scalar).                                     *}
{*****************************************************************}
procedure querynum(prompt        : prompt_string;
                   prompt_length : p_len;
                   min, max      : integer;
                 var answer      : integer;
                   units         : unit;
                   help_index    : integer);

   var range_violation,
         help_request,
          unit_change : boolean;
                    i : 1..max_prompt;
response, d_min, d_max : real;
                 dummy : char;

procedure getnumber;
   type pos = 1..max_num;
```

QueryNum Module

```
var         character : char;
     point, negative : boolean;
       number_string : packed array[1..max_num] of char;
    current_position,
    point_position,i : pos;

procedure show_number;
   var i : pos;
   begin
      goto_box(q,current_position+prompt_length+1,0);
      for i:=current_position to max_num do
          writech(number_string[i]);
      goto_box(q,current_position+prompt_length+1,0);
   end;

procedure sign(ch : char);
   begin
     goto_box(q,prompt_length+1,0);
     writech(ch);
     goto_box(q,current_position+prompt_length+1,0)
  end;

procedure add_digit;
   begin
      number_string[current_position]:=character;
      writech(character);
      if current_position<max_num
         then current_position:=current_position+1
         else writech(chr(cursorleft))
   end;

procedure insert_digit;
   var i : pos;
   begin
      if point
         then if point_position>=current_position
                 then begin
                         point_position:=point_position+1;
                         if point_position>max_num
                            then point:=false
                      end;
      if current_position<max_num
         then for i:=max_num downto current_position+1 do
                  number_string[i]:=number_string[i-1];
      number_string[current_position]:=' ';
      show_number
   end;

procedure delete_digit;
   var i : pos;
   begin
      if point
```

```
            then begin
                    if current_position=point_position
                       then point:=false;
                    if point_position>current_position
                       then point_position:=point_position-1;
                 end;
          if current_position<max_num
             then for i:=current_position to max_num-1 do
                      number_string[i]:=number_string[i+1];
          number_string[max_num]:=' ';
          show_number
       end;

    procedure convert;
    var power : real;

       procedure get_int_part(position : pos);
          var i : pos;
          begin
             power:=1;
             for i:=position downto 1 do
                 if ((number_string[i]>='0') and
                     (number_string[i]<='9'))
                    then begin
                            response:=response+power*
                                (ord(number_string[i])-ord('0'));
                            power:=power*10
                         end;
          end;

       procedure get_frac_part(position : pos);
          var i : pos;
          begin
             power:=0.1;
             for i:=position to max_num do
                 if ((number_string[i]>='0') and
                     (number_string[i]<='9'))
                     then begin
                             response:=response+power*
                                 (ord(number_string[i])-ord('0'));
                             power:=power/10
                          end;
          end;

       begin { convert }
          response:=0.0;
          if not point
             then get_int_part(max_num)
             else begin
                     if point_position>1
                        then get_int_part(point_position-1);
                     if point_position<max_num
```

```
                        then get_frac_part(point_position+1)
                end;
          if negative
             then response:=response*(-1);
       end;

    begin { getnumber }
       current_position:=1;
       for i:=1 to max_num do number_string[i]:=' ';
       show_number;
       help_request:=false; unit_change:=false; point:=false;
       negative:=false;
       repeat
          character:=getkey;
          case character of
            '0','1','2','3','4',
            '5','6','7','8','9',',' :
                  begin
                     if (point and (current_position=point_position))
                        then point:=false;
                     add_digit
                  end;
            '.' : if not point
                     then begin
                            point_position:=current_position;
                            add_digit; point:=true
                          end;
            '-' : begin
                     negative:=true;
                     sign('-')
                  end;
            '+' : begin
                     negative:=false;
                     sign(' ')
                  end;
            '?' : help_request:=true;
          end; { case }
          case ord(character) of
            ins : insert_digit;
            del : delete_digit;
         change : unit_change:=true;
      leftarrow : if current_position>1
                     then begin
                             current_position:=current_position-1;
                             writech(chr(cursorleft))
                          end;
     rightarrow : if current_position<max_num
                     then begin
                             current_position:=current_position+1;
                             writech(chr(cursorright))
                          end;
         return : convert;
```

```
         end { case }
      until ((character=chr(return)) or
             (character='?') or
             (character=chr(change)));
   end; { getnumber }

function to_inches(mils : integer): real;
   begin to_inches:=mils/1000.0 end;

function to_mm(mils : integer): real;
   begin to_mm:=to_inches(mils)*25.4 end;

begin { querynum }
   clear_box(q);
   repeat
      range_violation:=false;
      clear_line(q,0);
      for i:=1 to prompt_length do writech(prompt[i]);
      for i:=1 to max_num do writech(' ');
      case units of
         inches: begin
                    write('   inches     ');
                    d_min:=to_inches(min);
                    d_max:=to_inches(max);
                    clear_line(q,1);
                    write('Range: ',d_min:7:3,' to ',d_max:7:3)
                 end;
             mm: begin
                    write('   millimeters');
                    d_min:=to_mm(min);
                    d_max:=to_mm(max);
                    clear_line(q,1);
                    write('Range: ',d_min:7:3,' to ',d_max:7:3)
                 end;
           mils: begin
                    write('   mils       ');
                    d_min:=min; d_max:=max;
                    clear_line(q,1);
                    write('Range: ',d_min:6:0,' to ',d_max:6:0)
                 end;
         scalar: begin
                    d_min:=min; d_max:=max;
                    clear_line(q,1);
                    write('Range: ',d_min:6:0,' to ',d_max:6:0)
                 end;
      end; { case }
      getnumber;
      if unit_change
         then case units of
            scalar: units:=scalar;
              mils: units:=inches;
            inches: units:=mm;
```

QueryNum Module

```
                mm: units:=mils;
              end;
      if help_request
         then help(help_index);
      if ((not (unit_change or help_request)) and
          ((response<d_min) or (response>d_max)))
         then begin
                clear_line(q,2);
                write('Response not within range. Press any key to cont
inue');
                wait_key;
                clear_line(q,2);
                range_violation:=true
              end;
   until (not (help_request or range_violation or unit_change));
   case units of
      mils,scalar : answer:=round(response);
           inches : answer:=round(response*1000);
               mm : answer:=round(response*1000/2.54);
   end;
   clear_box(q);
end;

$odend.
```

QueryStr Module

```
{*****************************************************************}
{* String Query Module - Accepts string responses from the     *}
{*                       operator.                               *}
{*****************************************************************}
{* A Prompt supplied by the calling program is displayed to    *}
{* the user, along with the expected response length limits.   *}
{* All text typed by the user will be accepted as the          *}
{* response, terminated by a <RETURN>. All characters typed    *}
{* after MAX characters will be ignored, and <RETURN> will be  *}
{* ignored until MIN characters have been typed.               *}
{* The following editing features are available to the user:   *}
{* Character Insert - typing control-S will cause all          *}
{*                    characters to the right of the cursor    *}
{*                    to move to the right.  Any character at  *}
{*                    the last position will be lost if the    *}
{*                    string is already MAX characters long.   *}
{* Character Delete - typing control-D will cause all          *}
{*                    characters to the right of the cursor    *}
{*                    to move to the left, deleting the char-  *}
{*                    acter under the cursor.                  *}
{* Cursor Motion    - The left and right arrow keys (or their  *}
{*                    equivalents) will cause the cursor to    *}
{*                    move in the appropriate direction.  The  *}
{*                    cursor location when <RETURN> is pressed *}
{*                    is not important.  All visible charac-   *}
{*                    ters will be in the response.            *}
{* Other Features include specific help messages and UPPER or  *}
{* lower case conversion.                                      *}
{*****************************************************************}
module string_query;
const
    leftarrow = $08;  { Define arrow keys }
   rightarrow = $09;
   cursorleft = $18;  { Define cursor motion commands }
  cursorright = $19;
          ins = $13; {cntl S}   { Define Editing Commands }
          del = $04; {cntl D}
       return = $0D;
         null = $00;
          huh = $3F; { Question Mark '?' }
   max_prompt = 30;     { Maximum length of a prompt }
      max_str = 30;     { Maximum length of a string }

type
{ Query Type Definitions }
     prompt_string = packed array[1..max_prompt] of char;
             p_len = 1..max_prompt;
          position = 1..max_str;
       string_case = (upper, lower, none);
       char_string = packed array[1..max_str] of char;
             boxes = (q, m, h);
```

QueryStr Module

```
{ Screen I/O Declarations }
{ See the TermIO Module for details on the following }
external procedure goto_box(box : boxes; x,y : integer);
external procedure clear_box(box : boxes);
external procedure writech(ch : char);
external function getkey : char;

{ See the Help Module for details on the following }
external procedure help(help_index : integer);

{*****************************************************************}
{* Procedure : QuerySTR - Accept String response to query      *}
{* Parameters : Prompt, Prompt_Length, MIN, MAX, Answer,       *}
{*              String_Length, Make_Case, Help_Index           *}
{* Entry Conditions : Prompt is the question being asked of    *}
{* user, and is a fixed length string of characters.  Prompt_  *}
{* Length indicates how many of those characters are to be     *}
{* displayed.  MIN and MAX indicate the minimum and maximum    *}
{* number of characters that will be accepted as a response.   *}
{* Make_Case indicates what, if any, case conversion will be   *}
{* performed on return, and Help_Index identifies the help     *}
{* message associated with this query.                         *}
{* Exit Conditions : Answer will contain the text typed by the*}
{* user in response to Prompt.  The length of Answer will be   *}
{* between MIN and MAX characters, and will be indicated by    *}
{* String_Length.                                              *}
{* THe HELP routine will be invoked in response to '?' being   *}
{* typed at any time, and a question mark is therefore not a   *}
{* valid character in a response.                              *}
{*****************************************************************}
procedure querystr(prompt        : prompt_string;
                   prompt_length : p_len;
                   min, max      : position;
               var answer        : char_string;
               var string_length : position;
                   make_case     : string_case;
                   help_index    : integer);

   var help_request : boolean;
                  j : p_len;
                  i : position;

{*****************************************************************}
{* Procedure : GetString - Accept users response               *}
{* Process : Does most of the work - accepts and edits text    *}
{* supplied from the keyboard into Answer.                     *}
{*****************************************************************}
procedure getstring;
   var character : char;
    i, current_position : position;

{*****************************************************************}
```

QueryStr Module

```
{* Procedure : Redisplay                                          *}
{* Process : Display current version of Answer on the entry      *}
{* line.  The cursor position is not changed.                     *}
{*****************************************************************}
   procedure redisplay;
      var i : position;
      begin
         goto_box(q,prompt_length+current_position,0);
         for i:=current_position to string_length do
             writech(answer[i]);
         if string_length<max
            then for i:=string_length+1 to max do
                 writech('_');
         goto_box(q,prompt_length+current_position,0);
      end;

{*****************************************************************}
{* Procedure : Insert_Char - Open a space in Answer              *}
{* Process : The characters of Answer from the cursor to the     *}
{* end are shifted to the right one space.                       *}
{*****************************************************************}
   procedure insert_char;
      var i : position;
      begin
         if ((current_position < max) and
             (string_length>=current_position))
            then begin
                    for i:=max downto current_position+1 do
                        answer[i]:=answer[i-1];
                    if string_length<max
                       then string_length:=string_length+1;
                    answer[current_position]:=' ';
                    redisplay
              end;
      end;

{*****************************************************************}
{* Procedure : Delete_Char - Delete character under cursor       *}
{* Process : All the characters of Answer to the right of the    *}
{* cursor are moved right one space, and the last character      *}
{* replaced by a space.                                          *}
{*****************************************************************}
   procedure delete_char;
      var i : position;
      begin
         if ((current_position <max) and
             (string_length>=current_position))
            then begin
                    for i:=current_position to max-1 do
                        answer[i]:=answer[i+1];
                    string_length:=string_length-1;
                    answer[max]:=  ';
```

```
                    redisplay
                end;
        if ((current_position=max) and (string_length=max))
            then begin
                    string_length:=string_length-1;
                    answer[max]:=' ';
                    writech('_'); writech(chr(cursorleft))
                end;
    end;

{*****************************************************************}
{* Procedure : Move_Left                                         *}
{* Process : The cursor position within Answer is moved one      *}
{* space to the left, if possible.                               *}
{*****************************************************************}
  procedure move_left;
    begin
      if current_position>1
        then begin
                current_position:=current_position-1;
                writech(chr(cursorleft));
             end;
    end;

{*****************************************************************}
{* Procedure : Move_Right                                        *}
{* Process : The cursor position within Answer is moved one      *}
{* space to the right, if possible.                              *}
{*****************************************************************}
  procedure move_right;
    begin
      if current_position<max
        then begin
                current_position:=current_position+1;
                writech(chr(cursorright));
             end;
    end;

{*****************************************************************}
{* Procedure : Add_Char - append keyboard character to Answer    *}
{* Process : If there is room, Character will be added to        *}
{* Answer, and the cursor position will be updated.              *}
{*****************************************************************}
  procedure add_char;
    begin
      if string_length<current_position
        then string_length:=current_position;
      answer[current_position]:=character;
      writech(character);
      if current_position<max
        then current_position:=current_position+1
        else writech(chr(cursorleft));
```

```
      end;

    begin { getstring }
      goto_box(q,prompt_length+1,0);
      if max>max_str then max:=max_str;
      current_position:=1;
      string_length:=0;
      for i:=1 to max_str do answer[i]:=' ';
      help_request:=false;
      repeat
         character:=getkey;
         case ord(character) of
                ins : insert_char;
                del : delete_char;
          leftarrow : move_left;
         rightarrow : move_right;
             return : if string_length<min
                         then character:=chr(null);
                huh : help_request:=true;
         end;
         if ((character>=' ') and (character<=chr(127)))
            then add_char;
      until ((character=chr(return)) or help_request);
   end;

{*****************************************************************}
{* Procedure : Make_Upper - Convert all characters in Answer *}
{* Process : All alphabetic characters in Answer are trans-  *}
{* lated to upper case.  Punctuation and numbers are not     *}
{* effected.                                                 *}
{*****************************************************************}
procedure make_upper;
   var i : position;
   begin
      for i:=1 to max do
          if ((answer[i] >= 'a') and (answer[i] <= 'z'))
             then answer[i]:=chr(ord(answer[i])-ord('a')+ord('A'));
   end;

{*****************************************************************}
{* Procedure : Make_Lower - Convert all characters in Answer  *}
{* Process : All alphabetic characters in Answer are trans-  *}
{* lated to lower case.  Punctuation and numbers are not     *}
{* effected.                                                 *}
{*****************************************************************}
procedure make_lower;
   var i : position;
   begin
      for i:=1 to max do
          if ((answer[i] >= 'A') and (answer[i] <= 'Z'))
             then answer[i]:=chr(ord(answer[i])-ord('A')+ord('a'));
   end;
```

QueryStr Module

```
begin { querystr }
   clear_box(q);
   repeat
      goto_box(q,0,0);
      for j:=1 to prompt_length do writech(prompt[j]);
      writech(' ');
      for i:=1 to max do writech('_');
      goto_box(q,0,1);
      write('Response must be between ',min:2,' and ',max:2,' character
s');
      getstring;
      if help_request then help(help_index);
   until not help_request;
   case make_case of
        upper : make_upper;
        lower : make_lower;
   end;
   clear_box(q);
end;

modend.
{*****************************************************************}
{* Operating System Dependent File Operations                    *}
{*****************************************************************}
module dos_file;
type filespec = array[1..12] of char;
     drive_id = (A, B);

{See the Disk ID module for details on the following }
external procedure switchdisk(drive : drive_id; disk_id : integer);

{*****************************************************************}
{* Procedure : Run_File - Execute a program                      *}
{* Parameters : File_Name, Drive, Disk_ID                        *}
{* Entry Conditions : The parameters identify an executable      *}
{* program that is to be run.  Control will not return!!!!!      *}
{*****************************************************************}
procedure run_file(file_name : filespec; drive : drive_id;
                   disk_id : integer);
   var  f : file;
       fs : string[12];
        i : integer;
   begin
      switchdisk(drive,disk_id);
      i:=1;
      repeat
         fs[i]:=file_name[i];
          i:=i+1;
      until ((fs[i]=' ') or (i>12));
      fs[0]:=chr(i-1);
      assign(f,fs);
```

DOSFile Module

```
        reset(f);
        chain(f);
    end;

modend.
```

DiskID Module

```
{*****************************************************************}
{* DiskID Module - Identifies diskettes in drives and          *}
{*                 prompts operator when switches are          *}
{*                 necessary.                                  *}
{*****************************************************************}
{* Every diskette used by the system contains a small file     *}
{* with a DiskID number as its only element.  This file is     *}
{* referred to when the identity of any diskette is required.  *}
{* If the file isn't present, then the DiskID is set to 0.     *}
{*****************************************************************}
module diskid;
const max_prompt = 30;      { Maximum length of a prompt }
        max_str = 30;       { Maximum length of a string }

type
{ Query Type Definitions }
    prompt_string = packed array[1..max_prompt] of char;
            p_len = 1..max_prompt;
            yesno = (yes, no, i_dunno);
            boxes = (q, m, h);

{ File Access Type Definitions }
         drive_id = (A, B);
           id_num = file of integer;

{ Screen I/O Declarations }
{ See the TermIO Module for details on the following }
external procedure clear_line(box : boxes; y : integer);
external procedure wait_key;

{ Query Declarations }
{ See the QueryYN Module for details on the following }
external procedure query_yn(prompt        : prompt_string;
                            prompt_length : p_len;
                        var answer        : yesno;
                            help_index    : integer);

{ See the LinkFile Module for details on the following }
external procedure close_all(drive : drive_id);

{ CP/M BDOS call necessary to switch diskettes }
{ This function is defined in the MT+ Run-Time Library }
external function @bdos(func : integer; parm : word) : integer;

{*****************************************************************}
{* Procedure : Set_ID                                          *}
{* Parameters : Drive, ID_File                                 *}
{* Entry Conditions : ID_File is the identification file, and  *}
{* Drive identifies which drive is to be used.                 *}
{* Process : The physical filename 'DISK.ID' on drive Drive    *}
{* is associated with the pascal logical file ID_File.         *}
{*****************************************************************}
```

DiskID Module

```
procedure set_id(drive : drive_id; var id_file : id_num);
   begin
      case drive of
          A : assign(id_file,'A:DISK.ID');
          B : assign(id_file,'B:DISK.ID');
         else assign(id_file,'A:DISK.ID');
      end;
   end;

{*****************************************************************}
{* Function : WhichDisk                                          *}
{* Parameters : Drive                                            *}
{* Result : Disk_ID of diskette in Drive                         *}
{* Entry Conditions : Drive identifies the drive to be looked    *}
{* at.                                                           *}
{* Exit Conditions : The function will return the Disk_ID of     *}
{* the diskette in drive Drive.  A value of 0 indicates that     *}
{* either the diskette is not labelled, or there is no disk      *}
{* in the drive.                                                 *}
{*****************************************************************}
function whichdisk(drive : drive_id) : integer;
   var id_file : id_num;
            id : integer;
        status : byte;

   begin  { whichdisk }
      set_id(drive,id_file);
      { IORESULT is a function that returns the value of the CP/M
        BDOS result.  It is defined in the MT+ library. }
      reset(id_file); status:=ioresult;
      if status<>255
      then begin
             read(id_file,id);
             status:=ioresult;
           end;
      if status=0 then whichdisk:=id
                  else whichdisk:=0;
   end;

{*****************************************************************}
{* Procedure : ID_Disk - Write an ID number on the diskette      *}
{* Parameters : Drive, Disk_Num                                  *}
{* Entry Conditions : Drive identifies the drive the diskette    *}
{* is in, and Disk_Num is the number to be written in the        *}
{* ID_File.                                                      *}
{*****************************************************************}
procedure id_disk(drive:drive_id; disk_num:integer);
   var id_file : id_num;
       status  : integer;
   begin
      set_id(drive,id_file);
      rewrite(id_file);
```

```
      write(id_file,disk_num);
      close(id_file,status);
   end;

{*****************************************************************}
{* Procedure : SwitchDisk - Make sure that the right diskette *}
{* is in the right drive.                                        *}
{* Parameters : Drive, Disk_ID                                   *}
{* Entry Conditions : Drive and Disk_ID identify the desired  *}
{* setup; that is, Drive is to contain diskette Disk_ID.      *}
{* Exit Conditions : The proper diskette will have been moun- *}
{* ted in the proper drive.                                      *}
{*****************************************************************}
procedure switchdisk(drive : drive_id; disk_id : integer);
   var   answer : yesno;
            id : integer;
             d : char;

   begin { switchdisk }
      id:=whichdisk(drive);
      if id<>disk_id
        then begin
             if drive=A then d:='A' else d:='B';
             close_all(drive);
             repeat
              clear_line(q,-1);
              write('Insert disk number',disk_id:3,' into drive ',d:1);
              repeat
                 query_yn('Ready to Go?                    ',12,answer,100
);
              until answer=yes;
              clear_line(q,-1);
              id:=@bdos(13,wrd(0)); { Reset Disks }
              id:=whichdisk(drive);
              if id<>disk_id
                then begin
                      write('This is disk number',id:3,', not ',disk_id:
3);
                      write('! Press any key.'); wait_key;
                     end;
             until id=disk_id;
          end;
   end;

{*****************************************************************}
{* Procedure : New_Disk - Prompt the user to Label a New Disk *}
{* Parameters : Disk_ID                                          *}
{* Entry Conditions : Disk_ID is the number to be written on  *}
{* a newly formatted diskette.                                   *}
{*****************************************************************}
procedure new_disk(disk_id : integer);
   var answer : yesno;
```

DiskID Module

```
   begin
      clear_line(q,-2);
      write('Get out a blank formatted disk (disk 0 )');
      repeat
         query_yn('Are you ready?                    ',14,answer,19);
      until answer=yes;
      switchdisk(B,0);
      id_disk(B,disk_id);
      clear_line(q,-1);
      write('Label this disk as Disk # ',disk_id:3);
      repeat
      query_yn('Have you labelled it yet?      ',25,answer,22);
      until answer=yes;
      clear_line(q,-1);
      clear_line(q,-2);
   end;

modend.
```

LinkFile Module

```
{*****************************************************************}
{* LinkFile Module - Controls access to all files                *}
{*****************************************************************}
{* All system files are maintained as two doubly-linked          *}
{* circular lists.  One list contains all of the allocated       *}
{* records in the file, and the other contains all of the        *}
{* free records.  Every record in the file must belong to one    *}
{* of these lists.  Non linked files are also supported.         *}
{* For maximum portability, the maximum record length is 128     *}
{* bytes.                                                        *}
{*****************************************************************}
{$S+}
module file_access;
const
     end_of_list = -1;      { Invalid Record Number }
        max_open = 10;      { Maximum number of files }
    max_rec_size = 128;     { CP/M Maximum Record Size }
       max_buffs = 2;       { Maximum number of simultaneously open files
 }

type
{ File Access Type Definitions }
          drive_id = (A, B);
          filespec = packed array[1..12] of char;
         whichfile = 1..max_open;
             links = record
                        next,
                        prev : integer;
                     end;
           max_rec = record
                     case boolean of
                     true : (data : packed array[1..max_rec_size] of cha
r);
                     false: (link : links);
                     end;
         data_file = file of max_rec;
         file_desc = record
                           fs : string[14];
                       linked : boolean;
                        drive : drive_id;
                      on_line : boolean;
                      rec_len, { Record Length }
                      disk_id, { Diskette containing file }
                       status, { System Dependent File Status}
                        first, { Record Number of First Entry }
                    next_free, { Record Number of Free List }
                   recs_avail, { Number of Unused Records }
                    next_read, { Next record pointer }
                    last_read, { Last Record Accessed }
                   last_p_rec : integer; { Last Physical Record }
                      buf_num : byte;    { Current buffer number }
                     end;
```

```
            where = record
                      linked    : boolean;
                      file_name : filespec;
                      drive     : drive_id;
                      disk_id,
                      rec_len,
                      recs_avail,
                      first,
                      free      : integer;
                    end;

{ Global Arravs of file information }
var files  : array[whichfile] of file_desc;     { Logical file info }
    P_Buff : arrav[1..max_buffs] of data_file; { Phvsical buffer arrav }
    P_slot : array[1..max_buffs] of byte;       { Which slot buffer belon
gs to }
    next_1 : bvte;                              { Next slot tc free up }
    buffer : max_rec;                           { Temporary buffer area }

{ See the DiskIO Module for details on the following }
external procedure switchdisk(drive : drive_id; diskid : integer);

{ System Dependent Random Access File Routines }
procedure fread(file_num : whichfile; rec_num : integer;
                var buffer : max_rec; linkonly : boolean);

   var p_rec, rec_offset, i, bytes : integer;
   begin
      with files[file_num] do
      begin
         p_rec:=(rec_num * rec_len) div max_rec_size;
         rec_offset:=(rec_num * (rec_len mod max_rec_size)) mod max_rec_
size;
         if p_rec<>last_p_rec
            then begin
                    seekread(P_buff[buf_num],p_rec);
                    status:=ioresult;
                 end;
         if linkonlv then bytes:=4 else bytes:=rec_len;
         for i:=1 tc bytes do
         begin
            if rec_offset=max_rec_size
            then begin
                    rec_offset:=0;
                    p_rec:=p_rec+1;
                    seekread(P_buff[buf_num],p_rec);
                    status:=ioresult;
                 end;
            buffer.data[i]:=P_buff[buf_num].data[rec_offset+1];
            rec_offset:=rec_offset+1;
         end;
         last_p_rec:=p_rec;
```

```
      end;
   end;

procedure fwrite(file_num :whichfile; rec_num :integer;
                 var buffer : max_rec; linkonly : boolean);
   var p_rec, rec_offset, bytes,i : integer;

   begin
      with files[file_num] do
      begin
         p_rec:=(rec_num * rec_len) div max_rec_size;
         rec_offset:=(rec_num * (rec_len mod max_rec_size)) mod max_rec_
size;
         if p_rec<>last_p_rec then seekread(P_buff[buf_num],p_rec);
         if linkonly then bytes:=4 else bytes:=rec_len;
         for i:=1 to bytes do
         begin
            if rec_offset=max_rec_size
            then begin
                    seekwrite(P_buff[buf_num],p_rec);
                    p_rec:=p_rec+1;
                    seekread(P_buff[buf_num],p_rec);
                    rec_offset:=0;
                 end;
            P_buff[buf_num]^.data[rec_offset+1]:=buffer.data[i];
            rec_offset:=rec_offset+1;
         end;
         seekwrite(P_buff[buf_num],p_rec);
         status:=ioresult;
         last_p_rec:=p_rec;
      end;
   end;

{*****************************************************************}
{* Procedure : StateF - Determine Current Pointer Values       *}
{* Parameters : File_Num, First, Free                          *}
{* Entry Conditions : File_Num identifies the file to be       *}
{* looked at.                                                  *}
{* Exit Conditions : The current values of the heads of the    *}
{* allocated and free lists are assigned to First and Free.    *}
{*****************************************************************}
procedure statef(file_num:whichfile; var first,free : integer);
   begin
     first:=files[file_num].first;
     free:=files[file_num].next_free;
   end;

{*****************************************************************}
{* Function : RoomF - How much Room is Left in the File?       *}
{* Parameters : File_Num                                       *}
{* Result : Number of Unallocated Records in the file          *}
{*****************************************************************}
```

LinkFile Module

```
function roomf(file_num : whichfile) : integer;
   begin
      roomf:=files[file_num].recs_avail;
   end;

{*****************************************************************}
{* Procedure : ResetF - Set pointers to the top of the file    *}
{* Parameters : File_Num                                         *}
{* Process : The file pointers Next_Read and Last_Read are     *}
{* set to the first allocated record in the file.  Any file    *}
{* access after a resetf will access the first record.         *}
{*****************************************************************}
procedure resetf(file_num : whichfile);
   begin
      with files[file_num] do
      if linked
         then begin
                 last_read:=first;
                 next_read:=first;
              end
         else next_read:=0;
   end;

{*****************************************************************}
{* Procedure : InitF - Initialize a slot in the Files array    *}
{* Parameters : File_Num, File_Loc                               *}
{* Entry Conditions : File_Num identifies the slot to be       *}
{* initialized, and File_Loc contains the initialization       *}
{* parameters.                                                   *}
{* Process : The File_Num slot will be loaded with File_Loc    *}
{* and the file will be set to off_line status.  The file      *}
{* will then be reset to the first record.                       *}
{*****************************************************************}
procedure initf(file_num : whichfile; file_loc : where);
   var i : 1..12;
       d : string[2];
     tfs : string[12];
   begin
      with files[file_num] do
      begin
         if file_loc.drive=A then d:='A:' else d:='B:';
       { Convert character array file_name into string }
         i:=1;
         while file_loc.file_name[i]<>' ' do
         begin
            tfs[i]:=file_loc.file_name[i];
            i:=i+1
         end;
         tfs[0]:=chr(i-1);
       { Add drive designation to filename }
         fs          :=concat(d,tfs);
         linked      := file_loc.linked;
```

```
         disk_id    := file_loc.disk_id;
         drive      := file_loc.drive;
         rec_len    := file_loc.rec_len;
         recs_avail := file_loc.recs_avail;
         first      := file_loc.first;
         next_free  := file_loc.free;
         on_line    := false;
      end;
      resetf(file_num);
   end;

{*****************************************************************}
{* Procedure : Init_Files -Onetime Files Array Initialization *}
{* Process : Sets ALL file slots off-line so that a Close_All *}
{* operation (See Below) will not attempt to close file slots *}
{* that were never initialized. Should only be called once.   *}
{*****************************************************************}
procedure init_files;
   var i : 1..max_open;
   begin
      for i:=1 to max_open do
          with files[i] do
          begin
             on_line:=false;
             fs[0]:=chr(0);
             last_p_rec:=-1;
          end;
      for i:=1 to max_buffs do P_slot[i]:=0;
      next_1:=1;
   end;

{*****************************************************************}
{* Procedure : CloseF - Close Random Access File               *}
{* Parameters : File_Num                                       *}
{* Process : The file in slot File_Num is closed and the On-   *}
{* Line flag is reset. If already off-line, nothing is done.   *}
{*****************************************************************}
procedure closef(file_num : whichfile);
   begin
      with files[file_num] do
      begin
      if on_line
         then begin
                close(P_buff[buf_num],status);
                on_line:=false;
                last_p_rec:=-1;
                P_slot[buf_num]:=0;  { Release the buffer }
              end;
      end;
   end;

{*****************************************************************}
```

LinkFile Module

```
{* Procedure : OpenF - Open file for reading or writing       *}
{* Parameters : File_Num                                      *}
{* Process : The file in slot File_Num is brought on-line.    *}
{* If the file is already On-Line, then nothing need be done. *}
{* If, however, the file is off-line, then the diskette       *}
{* containing the file must be mounted, the system dependent  *}
{* association between logical and physical files must be made*}
{* and the file opened for random access.                     *}
{************************************************************}
procedure openf(file_num : whichfile);
  var i : integer;
  begin
     with files[file_num] do
        if not on_line then
           begin
              switchdisk(drive,disk_id);
              { Look for an available buffer }
              buf_num:=0; i:=1;
              repeat
                if P_slot[i]=0 then buf_num:=i;
                i:=i+1;
              until ((i>max_buffs) or (buf_num<>0));
              { If one isn't available, Free one up }
              if buf_num=0
                 then begin
                         closef(P_slot[next_1]);
                         buf_num:=next_1;
                         next_1:=next_1+1;
                         if next_1>max_buffs then next_1:=1;
                      end;
              { Reserve the Buffer for this slot }
              P_slot[buf_num]:=file_num;
              { Attempt to open the file }
              open(P_buff[buf_num],fs,i);
              { A return code of 255 means the file doesn't exist,
                so it must be created. }
              if i=255
                 then begin
                         { Create the file }
                         assign(P_buff[buf_num],fs);
                         rewrite(P_buff[buf_num]);
                         close(P_buff[buf_num],i);
                         { Open the newly created file }
                         open(P_buff[buf_num],fs,i);
                      end;
              on_line:=true;
           end;
  end;

{************************************************************}
{* Procedure : Close_All - Close all files on a drive        *}
{* Parameters : Drive                                        *}
```

LinkFile Module

```
{* Entry Conditions : Drive indicates which drive is to have  *}
{* all of its open files closed.                              *}
{* Process : This routine is called prior to removal of a disk*}
{* from a drive to  ensure file integrity.  Each slot in Files*}
{* is checked to see if the drive matches.  If it does, the   *}
{* file is closed.                                            *}
{************************************************************}
procedure close_all(drive : drive_id);
   var fx : whichfile;
   begin
      for fx:=1 to max_open do
        if files[fx].drive=drive then closef(fx)
   end;

{************************************************************}
{* Procedure : ReadF - Read the Next Record                   *}
{* Parameters : File_Num, Buffer, EOLF                        *}
{* Entry Conditions : File_Num identifies to file to read.    *}
{* Next_Read contains the record number of the next record to *}
{* be accessed.                                               *}
{* Exit Conditions : Buffer contains the record read, and     *}
{* Next_Read and Last_Read will be updated accordingly, unless*}
{* an end of file condition was detected.  In this case, the  *}
{* Buffer will not be modified and EOLF will be set.          *}
{************************************************************}
procedure readf(file_num : whichfile; var buffer : max_rec;
                var eolf : boolean);
   var i    : integer;
       link : links;
   begin
      openf(file_num);
      with files[file_num] do
      if linked
        then begin
                if ((next_read=first) and (last_read<>first)) or
                    (next_read=end_of_list)
                then eolf:=true
                else begin
                        fread(file_num,next_read,buffer,false);
                        last_read:=next_read;
                        next_read:=buffer.link.next;
                        if last_read=next_read { There is only one recor
d }
                           then eolf:=true
                           else eolf:=false;
                     end;
             end
        else begin
                fread(file_num,next_read,buffer,false);
                next_read:=next_read+1;
                if status=0
                   then eolf:=false
```

```
                    else eolf:=true;
                end;
    end; { readf }

{*****************************************************************}
{* Procedure : WriteF - Update the Last_Record accessed        *}
{* Parameters : File_Num, Buffer, WriteOK                      *}
{* Entry Conditions : File_Num identifies the file to write    *}
{* to.  Buffer contains the information to be written.  Last_ *}
{* Read is the record number to be written to.                 *}
{* Exit Conditions : Buff will have been written to the file. *}
{* No re-positioning will occur, so successive writes will     *}
{* over-write the same record.  WriteOK will be TRUE if there *}
{* were no write errors.                                       *}
{*****************************************************************}
procedure writef(file_num : whichfile; var buffer : max_rec;
                 var writeok : boolean);
    var i : 1..max_rec_size;
    begin
       openf(file_num);
       with files[file_num] do
        if linked
           then if last_read<>end_of_list
                   then begin
                           writeok:=true;
                           fread(file_num,last_read,buffer,true);
                           fwrite(file_num,last_read,buffer,false);
                        end
                   else writeok:=false
           else begin
                   fwrite(file_num,next_read,buffer,false);
                   next_read:=next_read+1;
                   if status=0
                      then writeok:=true
                      else writeok:=false;
                end;
    end; { writef }

{*****************************************************************}
{* Function : Del_Rec - Delete a record from one of the lists *}
{* Parameters : File_Num, Pointer                              *}
{* Result : Record number of deleted record                    *}
{* Entry Conditions : File_Num indicates file to use, Pointer *}
{* is the record number to be deleted.                         *}
{* Exit Conditions : The record will be deleted from the list *}
{* and pointer will be set to the the next record in the list.*}
{*****************************************************************}
function del_rec(file_num : whichfile; var pointer : integer) : integer;
    var link : links;
    begin
       openf(file_num);
       with files[file_num] do
```

```
      begin
         del_rec:=pointer;
         { Read the Next and Previous Record numbers }
         fread(file_num,pointer,buffer,true);
         link:=buffer.link;
         if link.next=pointer { It's the last record }
            then pointer:=end_of_list
            else begin
                 { Make the Next record the new Current record }
                   pointer:=link.next;
                 { Delete the record from the forward list }
                   fread(file_num,link.prev,buffer,true);
                   buffer.link.next:=link.next;
                   fwrite(file_num,link.prev,buffer,true);
                 { Delete the record from the backward list }
                   fread(file_num,link.next,buffer,true);
                   buffer.link.prev:=link.prev;
                   fwrite(file_num,link.next,buffer,true);
                 end;
      end; { with }
   end; { del rec }

{*****************************************************************}
{* Procedure : Ins_Rec - Insert a record into one of the lists*}
{* Parameters : File_Num, New_Rec, Pointer                       *}
{* Entry Conditions : File_Num indicates which file to use,    *}
{* New_Rec is the number of the record to be inserted into the*}
{* list, and Pointer is the number of the record that New_Rec *}
{* will be inserted after.                                       *}
{* Exit Conditions : The links will have been adjusted so that*}
{* New_Rec logically follows Pointer in the list.              *}
{*****************************************************************}
procedure ins_rec(file_num : whichfile; new_rec : integer;
                  pointer : integer);
   var link : links;
   begin
     openf(file_num);
     with files[file_num] do
      begin
       if pointer=end_of_list  { Empty List }
          then begin
             { Make New_Rec the only entry in the list }
               buffer.link.prev:=new_rec;
               buffer.link.next:=new_rec;
               fwrite(file_num,new_rec,buffer,true);
             end
       else begin
          { Insert New_Rec into the forward list }
            fread(file_num,pointer,buffer,true);
            link.prev:=pointer; link.next:=buffer.link.next;
            buffer.link.next:=new_rec;
            fwrite(file_num,pointer,buffer,true);
```

```
            { Insert New_Rec into the backward list }
               fread(file_num,link.next,buffer,true);
               buffer.link.prev:=new_rec;
               fwrite(file_num,link.next,buffer,true);
            { Make New_Rec point to its neighbors }
               buffer.link:=link;
               fwrite(file_num,new_rec,buffer,true);
             end;
      end; { with }
   end; { ins_rec }

{*************************************************************}
{* Procedure : InsertF - Insert a record into the file       *}
{* Parameters : File_Num, Rec_Num, Buffer                    *}
{* Entry Conditions : File_Num indicates the file to use.    *}
{* Buffer contains the information to be inserted.  Last_    *}
{* Record is the number of the record which Buffer is to be  *}
{* inserted after.                                           *}
{* Exit Conditions : Rec_Num will be the record number which *}
{* was assigned to Buffer in the file.  Once a record is added*}
{* to the file, its position will never change.              *}
{*************************************************************}
procedure insertf(file_num : whichfile; var rec_num : integer;
                  var buffer : max_rec);
   begin
      openf(file_num);
      with files[file_num] do
      begin
         if next_free<>end_of_list
         then begin { There IS a free record }
                recs_avail:=recs_avail-1;
               { Delete a record from the free list }
                rec_num:=del_rec(file_num,next_free);
               { And add it to the allocated list }
                ins_rec(file_num,rec_num,last_read);
               { Update the record pointers }
                last_read:=rec_num;
                if first=end_of_list
                then begin {  This is the first record }
                       next_read:=last_read;
                       first:=last_read;
                     end;
               { Write Buffer to the file }
                fread(file_num,last_read,buffer,true);
                fwrite(file_num,last_read,buffer,false);
              end;
      end; { with }
   end; { insertf }

{*************************************************************}
{* Procedure : DeleteF - Delete a record from a file         *}
{* Parameters : File_Num                                     *}
```

LinkFile Module

```
{* Entry Conditions : File_Num is the file to be used.  Last_ *}
{* Read is the record to be deleted.                          *}
{* Exit Conditions : Last_Read will point to the record foll- *}
{* owing the deleted one.                                     *}
{
*************************************************************}
procedure deletef(file_num : whichfile);
   var rec_num : integer;
   begin
      with files[file_num] do
        begin
           if first<>end_of_list
              then begin { There IS a record to delete }
                   recs_avail:=recs_avail+1;
                 { Delete the record from the file }
                   rec_num:=del_rec(file_num,last_read);
                   if last_read=end_of_list
                      then begin
                              first:=end_of_list;
                              next_read:=end_of_list;
                           end;
                 { Add the deleted record to the free list }
                   ins_rec(file_num,rec_num,next_free);
                   next_free:=rec_num;
                end;
        end;
   end;

{*************************************************************}
{* Procedure : PosF - Position file pointers                  *}
{* Parameters : File_Num, Rec_Num                             *}
{* Entry Conditions : File_Num indicates the slot to use, and *}
{* Rec_Num is the record number to which the file pointers    *}
{* will be set.                                               *}
{*************************************************************}
procedure posf(file_num : whichfile; rec_num : integer);
begin
   with files[file_num] do
   begin
      if ((rec_num=first) and (not linked))
      then resetf(file_num)
      else next_read:=rec_num;
   end;
end;

{*************************************************************}
{* Procedure : CreateF - Create a new Index for the file      *}
{* Paramaters : File_Num, How_Many                            *}
{* Entry Conditions : File_Num indicates which file to use,   *}
{* and How_Many indicates how many records to allocate to the *}
{* file.                                                      *}
{* Process : The links will be initialized as follows : all of*}
```

LinkFile Module

```
(* the records will be assigned to the Free list, and the     *)
(* allocated list will be empty.  None of the information in  *)
(* the file will be erased, but it will not be accessable.    *)
(************************************************************)
procedure createf(file_num : whichfile; how_many : integer);
   var rec : integer;
   begin
      openf(file_num);
      with files[file_num] do
      begin
         first:=end_of_list; next_free:=0;
         recs_avail:=how_many;
         for rec:=0 to how_many-1 do
         begin
            { Set up the forward and backward links }
            buffer.link.next:=rec+1; buffer.link.prev:=rec-1;
            { Make the next record of the Last entry point
              to the First entry }
            if buffer.link.next=how_many then buffer.link.next:=0;
            { Make the previous record of the First entry
              point to the Last entry - a circular list. }
            if buffer.link.prev=-1 then buffer.link.prev:=how_many-1;
            fwrite(file_num,rec,buffer,true);
         end;
      end;
      closef(file_num);
   end;

procedure erasef(file_num : whichfile);
   begin { erasef }
     openf(file_num);
     purge(P_buff[files[file_num].buf_num]);
   end;

modend.
```

Menu Module

```
{*****************************************************************}
{* Menu Module - procedures for the manipulation of menus      *}
{*****************************************************************}
{* Menus may either be read directly from a file, or created   *}
{* dynamically.  In either case, the format of the menu in     *}
{* memory is identical. The selection of an item by the user   *}
{* is done by moving the cursor next to the desired item and   *}
{* pressing <RETURN>.  Help is available for each item by      *}
{* pressing <?>.  The menu structure allows for more than one  *}
{* level.  If the item selected has sub-selections, control    *}
{* will not return to the user until a terminal item is selec-*}
{* ted.                                                        *}
{*****************************************************************}
module menu_module;
const
      uparrow = $5B;
    downarrow = $0A;
   cursorleft = $18;  { Define cursor motion commands }
       return = $0D;
       escape = $03; {cntl C}
         null = $00;
          huh = $3F;
     max_item = 20;      { Maximum length of a menu item }
        menus = 2;       { Menu file is file number two }
type
{ Menu Type Definitions }
      menu_string = packed array[1..max_item] of char;
         item_ptr = ^menu_item;
     { In Memory Menu structure }
        menu_item = record
                      item_text : menu_string; {Text User sees}
                      item_code,               {Code returned }
                      help_index : integer;
                      next_item,               { Pointer }
                      previous_item,           { Pointer }
                      next_level : item_ptr;   { Pointer }
                    end;
     { Menu File Record Structure }
         menu_In = record
                      menu_number : integer;  { Menu ID }
                      bump_down,        { True if item has
                                          Sub-selections }
                      bump_up : boolean;{ True if last item of
                                          current level }
                      item_text : menu_string;
                      item_code,
                      help_index : integer;
                    end;
       whichfile = 1..10;
           boxes = (q, m, h);     { Query, Menu, or Help Box }

{ Define current menu environment }
```

Menu Module

```
var old        : integer;    { Last menu accessed }
    first_item : item_ptr;   { Top of last menu }
    top_list   : item_ptr;   { Top of Current List }
    list       : item_ptr;   { Current position inside list }
    count      : integer;    { Number of lines in current level }
    max_lines  : integer;    { Number of lines allowed in a menu }

{ See TermIO Module for details on the following }
{ Screen I/O Declarations }
external procedure goto_box(box : boxes; x,y : integer);
external procedure clear_line(box : boxes; y : integer);
external procedure clear_box(box : boxes);
external function get_count(box : boxes) : integer;
external procedure writech(ch : char);
external function getkey : char;

{ See the LinkFile Module for details on the following }
external procedure resetf(file_num : whichfile);
external procedure readf(file_num : whichfile; var buffer : menu_ln;
                         var eolf : boolean);

{ See the Help Module for details on the following }
external procedure help(help_index:integer);

{*****************************************************************}
{* Procedure : Erase_Menu - Delete a menu structure from Heap *}
{* Parameters : First_Item                                       *}
{* Entry Conditions : First_Item points to the first item of  *}
{* the menu structure currently defined.                         *}
{* Process : Each item of the current level will be Disposed. *}
{* If the menu menu item has a sub-level, however, Erase_Menu *}
{* will be recursively called to erase that level first.      *}
{* This will insure that the entire tree structure is erased. *}
{*****************************************************************}
procedure erase_menu(first_item : item_ptr);
   var current_item : item_ptr;
   begin
     { Make sure there is something to erase! }
      if first_item<>nil
      then repeat
            { Erase all lower levels of menu }
              if first_item^.next_level<>nil
                 then erase_menu(first_item^.next_level);
              current_item:=first_item;
            { Point to the next item }
              first_item:=first_item^.next_item;
              dispose(current_item);
           until first_item=nil;
   end;

{*****************************************************************}
{* Function : Read_Menu - Read a Menu from the Menu File      *}
```

Menu Module

```
{* Parameters : Menu_Number                                          *}
{* Result : Pointer to first item of Menu                            *}
{* Entry Conditions : Menu_Number identifies which menu to           *}
{* read.                                                             *}
{* Exit Conditions : The menu will have been read from the           *}
{* Menu File into memory.  The result of the function is a           *}
{* pointer to the first item of the menu.                            *}
{*******************************************************************}
function read_menu(menu_number : integer) : item_ptr;
   var menu_line : menu_ln;    { A Line from the file }
            item : item_ptr;
            eolf : boolean;

{*******************************************************************}
{* Procedure : Read_Level - Read all menu items at current          *}
{*                          level into memory                        *}
{* Parameters : Last_Item, Last_Line                                 *}
{* Global Variables : EOLF                                           *}
{* Entry Conditions : Last_Item points to an allocated but           *}
{* not yet initialized menu item.  Last_Line is the last item        *}
{* read from the file. EOLF is the end of file indicator.            *}
{* Process : First, the values in Last_Line are assigned to          *}
{* Last_Item.  Then, a test is made to see if there are sub-         *}
{* items under this one. If there are, then Read_Level is            *}
{* called recursively to read it.  Finally, a test is made to        *}
{* see if this item is the last item at this level.  If it is,       *}
{* then control will return to the caller.                           *}
{*******************************************************************}
   procedure read_level(last_item : item_ptr;
                        last_line : menu_ln);
      var level,item : item_ptr;
          menu_line : menu_ln;
               done : boolean;
      begin
         done:=false;
         repeat
           { Assign File values to Memory Menu }
            last_item^.item_text:=last_line.item_text;
            last_item^.item_code:=last_line.item_code;
            last_item^.help_index:=last_line.help_index;
            if last_line.bump_down
               then begin { There ARE sub-items }
                     { Allocate a new item and point to it }
                       new(level); last_item^.next_level:=level;
                     { Make it the first item of the next level}
                       level^.previous_item:=nil;
                     { Pre-read the Menu File }
                       readf(menus,menu_line,eolf);
                     { Read in the rest of this level }
                       read_level(level,menu_line);
                    end
               else last_item^.next_level:=nil;
```

```
            if (eolf or last_line.bump_up)
               then begin { Done with this level }
                       last_item^.next_item:=nil;
                       done:=true
                    end
               else begin
                     { Read in the next item for this level }
                       readf(menus,menu_line,eolf);
                     { Allocate storage and point to it }
                       new(item); last_item^.next_item:=item;
                       item^.previous_item:=last_item;
                     { Set up for next iteration }
                       last_item:=item; last_line:=menu_line
                    end;
         until done;
      end;

   begin { read_menu }
      resetf(menus);
    { First, find the appropriate Menu in the Menu File }
      repeat
         readf(menus,menu_line,eolf);
      until menu_line.menu_number=menu_number;
    { Allocate storage for and point to the first item }
      new(item); item^.previous_level:=nil;
      read_menu:=item;
    { Read in the entire menu structure }
      read_level(item,menu_line);
  end;

{*****************************************************************}
{* Function : Select_Menu_Item                                   *}
{* Parameters : Menu_Index                                       *}
{* Result : Item_Code of the selected terminal item.             *}
{* Entry Conditions : Menu_Index points to the first item of     *}
{* a Menu currently in memory.                                   *}
{* Exit Conditions : The user will have selected one of the      *}
{* terminal menu items (an item with no sub-items).  Its code    *}
{* will be returned as the value of the function.  If a non-     *}
{* terminal item is selected, then a recursive call to this      *}
{* function will be made, until a terminal item is selected.     *}
{*****************************************************************}
function select_menu_item(menu_index : item_ptr) : integer;
   var index     : item_ptr;     { Current Menu Item }
       character : char;         { Keyboard entry Character }
       selection : integer;      { Selection code }
       l         : integer;      { Current line number in menu box }

{*****************************************************************}
{* Procedure : Menu_Display - Display all Menu choices at the    *}
{*                            current level                      *}
{* Process : The menu box area of the screen is cleared, and     *}
```

Menu Module

```
{* the choices available at the current level of the menu are *)
{* displayed on sequential lines of the screen.  The cursor is*)
{* positioned to the left of the first item.                   *)
{*************************************************************}
   procedure menu_display;
      var index : item_ptr;
            l : integer;
      begin
         clear_box(m);
         l:=0;
         index:=menu_index;
         repeat
          { Indent each item to leave room for the cursor }
            goto_box(m,2,l);
            write(index^.item_text);
          { Point to the next item at this level }
            l:=l+1; index:=index^.next_item;
         until index=nil;
         goto_box(m,0,0);
         writech( *'); writech(chr(cursorleft));
      end;

   begin { Select Menu Item }
      menu_display;
      index:=menu_index;
      l:=0;
      repeat
        character:=getkey;
        case ord(character) of
            uparrow : if index^.previous_item<>nil
                      then begin
                             writech(' '); writech(chr(cursorleft));
                             l:=l-1; goto_box(m,0,l);
                             writech('*'); writech(chr(cursorleft));
                             index:=index^.previous_item
                           end;
          downarrow : if index^.next_item<>nil
                      then begin
                             writech(' ');writech(chr(cursorleft));
                             l:=l+1; goto_box(m,0,l);
                             writech('*'); writech(chr(cursorleft));
                             index:=index^.next_item
                           end;
                huh : begin
                             help(index^.help_index);
                             menu_display;
                             writech(  '); writech(chr(cursorleft));
                             goto_box(m,0,l);
                             writech('*'); writech(chr(cursorleft));
                      end;
             return : if index^.next_level=nil
                      then select_menu_item:=index^.item_code
```

```
                        else begin  { recursive call }
                                selection:=select_menu_item(index^.next_le
vel);
                                if selection=0
                                then begin
                                 { Return from lower level with
                                   no selection, so redisplay
                                   current level }
                                   character:=chr(null);
                                   menu_display;
                                   index:=menu_index
                                 end
                                else select_menu_item:=selection;
                              end;
                escape : select_menu_item:=0;
           end; { case }
        until ((character=chr(return)) or (character=chr(escape)));
        clear_box(m);
     end; { select_menu_item }

{*****************************************************************}
{* Function : Menu - Make a selection from a File Menu          *}
{* Parameters : Current                                         *}
{* Global Variables : Old, First_Item                           *}
{* Result : Item_Code of selected terminal item.                *}
{* Entry Conditions : Current is the number of the menu to be *}
{* made the new current menu.  Old is the number of the last  *}
{* menu accessed, and First_Item points to the old menu.        *}
{* Exit Conditions : Old is updated to Current, and the         *}
{* function returns the selected code.                          *}
{*****************************************************************}
function menu(current : integer) : integer;
   begin
      if old <> current
      then begin { A new menu must be read from the file }
               erase_menu(first_item);
               first_item:=read_menu(current);
               old:=current;
             end;
      menu:=select_menu_item(first_item);
   end;

{*****************************************************************}
{* Procedure : Init_List - Initialize a Memory List to empty  *}
{* Process : A list is to be built in memory.  It must first  *}
{* be initialized.  Any old list will be erased, and the size *}
{* of the list displayed will be set to fill the menu-box.    *}
{*****************************************************************}
procedure init_list;
   begin
      erase_menu(top_list);
      top_list:=nil; list:=nil; count:=0;
```

```
    { Get the size of the Menu Box }
      max_lines:=get_count(m);
   end;

{*****************************************************************}
{* Procedure : Build_List - Add an item to the Memory List     *}
{* Parameters : Item_Text, Item_Code                           *}
{* Global Variables : Top_List, List, Count, Max_Lines         *}
{* Entry Conditions : Item_Text and corresponding Item_Code    *}
{* are to be added to the list in memory.                      *}
{* Process : The item will be added to the list.  If the       *}
{* number of items in the current level exceeds the capacity   *}
{* of the menu-box, then a new level will be created.  Count   *}
{* maiantains the number of items in the current level, and    *}
{* Max_Lines is the size of the menu-box.  Both are            *}
{* initialized by Init_List. The first call to this procedure  *}
{* will define Top_List, and position for subsequent calls     *}
{* will be maintained by List.                                 *}
{*****************************************************************}
procedure build_list(item_text:menu_string; item_code:integer);
   var entry : item_ptr;
         c : integer;
   begin
      new(entry);
      if list<>nil
      then begin  { This isn't the first element of the list }
             list^.next_item:=entry;
             entry^.previous_item:=list;
           end
      else begin  { This IS the first element of the list }
             entry^.previous_item:=nil;
             top_list:=entry;
           end;
      if count=(max_lines-1)
      then begin  { Start a new level of menu }
             entry^.next_item:=nil;
             new(list);
             list^.previous_item:=nil;
             list^.next_level:=nil;
             entry^.next_level:=list;
             count:=0;
             entry^.item_text:='Rest of the list    ';
             entry^.help_index:=17;
           end
      else begin  { Add to present level of menu }
             entry^.next_level:=nil;
             list:=entry;
           end;
      list^.item_text:=item_text;
      list^.item_code:=item_code;
      list^.help_index:=18;
      list^.next_item:=nil;  { In case this is the last one }
```

```
      count:=count+1;
   end; { build list }

{*****************************************************************}
{* Function : Select_List - Select an item from Memory List    *}
{* Global Variables : Top_List                                 *}
{* Result : Item_Code of selected item.                        *}
{* Process : After a list hes been built in memory through     *}
{* calls to Build_List, an item may be selected with this      *}
{* function.  Since the Structure of both lists and menus is   *}
{* identical, Select_Menu_Item is used to do the actual work.  *}
{*****************************************************************}
function select_list : integer;
   begin
      select_list:=select_menu_item(top_list);
   end;

{*****************************************************************}
{* Procedure : Init_Menu - Initialize Global Variables         *}
{* Global Variables : Old, First_Item, Top_List                *}
{* Process : Prior to the first use of either a menu or a      *}
{* list, the pointers must be initialized.                     *}
{*****************************************************************}
procedure init_menu;
   begin
      old:=0; first_item:=nil; top_list:=nil;
   end;

modend.
```

Help Module

```
{*****************************************************************}
{* Help Module - Provides information to the user about the    *}
{* system.  The help messages are stored in the Help File, and*}
{* are indexed.  Each query or menu item has been assigned a  *}
{* help index, which is used to look up the associated help   *}
{* information.                                                *}
{*****************************************************************}
{$S+}
module helper;
const max_help = 30;     { Maximum length of a help message line}
      helps = 1;         { Help file is file number one }

type help_msg = record
                  help_index : integer;
                  line : packed array[1..max_help] of char;
                end;
     whichfile = 1..10;
         boxes = (q, m, h);

{ See the TermIO Module for details on the following }
{ Screen I/O Declarations }
external procedure clear_line(box : boxes; y : integer);
external procedure waitkey;

{ See the LinkFile Module for details on the following }
{ File Access Declarations }
external procedure readf(file_num : whichfile; var buffer : help_msg;
                         var eolf : boolean);
external procedure resetf(file_num : whichfile);

{*****************************************************************}
{* Procedure : Help                                            *}
{* Parameters : Help_Index                                     *}
{* Entry Conditions : Help_Index identifies which help message*}
{* to look up.                                                 *}
{* Process : The Help File is scanned until a matching index  *}
{* is found.  This line and all subsequent lines are displayed*}
{* in the help-box area of the screen, until a line with a    *}
{* different index is found.                                   *}
{*****************************************************************}
procedure help(help_index : integer);
   var mess : help_msg;
       eolf : boolean;
       l    : integer;
   begin
      resetf(helps);
    { Find the first line of the help message, if there is one.}
      repeat
         readf(helps,mess,eolf);
      until (mess.help_index=help_index) or eolf;
      l:=0;
    { Display all lines with matching index numbers }
```

```
      while ((not eolf) and (mess.help_index=help_index)) do
         begin
            clear_line(h,l);
            write(mess.line); l:=l+1;
            readf(helps,mess,eolf);
         end;
      clear_line(h,l);
      write('End of Help - Press any Key.');
      waitkey;
   end;

modend.
```

Users Module

```
{*****************************************************************}
{* Users Module - Maintains the List of Users for the Command *}
{*                Processor                                      *}
{*****************************************************************}
module user_file;
const max_str = 30;      { Maximum length of a string }
    user_file = 3;
type
char_string = packed array[1..max_str] of char;
  whichfile = 1..10;
      links = record
                next,
                prev : integer;
              end;

{ List of Users File Type Definitions }
{ All the information maintained about a user }
   user_entry = record
                  link : links;
                  name : char_string;
                    id : integer;
              password : char_string;
               end;

{ Global Variable initialized by Read_Args routine. See
  the Argument Module for details. }
var next_user : external integer;

{ File Access Declarations }
{ See the LinkFile Module for details on the following }
external procedure readf(file_num : whichfile; var buffer : user_entry;
                         var eolf : boolean);
external procedure insertf(file_num : whichfile ; var rec_num : integer;
                           var buffer : user_entry);
external procedure resetf(file_num : whichfile);
external function roomf(file_num : whichfile) : integer;

{*****************************************************************}
{* Function : Lookup                                             *}
{* Parameters : User_Name, Password, ID                          *}
{* Entry Conditions : User_Name contains the name of the user    *}
{* as he typed it in.                                            *}
{* Process : User_Name is looked up in the User_File             *}
{* Exit Conditions : If the name is found, then Password and     *}
{* ID are set to the matching entries in the file, and the       *}
{* function returns a TRUE value.  If the name isn't in the      *}
{* list, the function returns a FALSE value.                     *}
{*****************************************************************}
function lookup(user_name : char_string;
                var password : char_string;
                var id : integer) : boolean;
   var user_info : user_entry;     { File entry for comparison }
```

Users Module

```
            eolf : boolean;        { End of List of Users }
   begin
      lookup:=false;   { Haven't found him yet }
      resetf(user_file);
      repeat
         readf(user_file,user_info,eolf);
         if user_info.name=user_name
            then begin  { This user IS present in the file }
                    password:=user_info.password;
                    id:=user_info.id;
                    lookup:=true { We found him }
                 end;
      until (eolf or (user_info.name=user_name));
   end;

{*****************************************************************}
{* Function : NewUserOK                                          *}
{* Entry Conditions : An Inquiry is being made to see if there*}
{* is room in the List of Users for another entry.               *}
{* Process : The RoomF function is called to see if there is  *}
{* any space available.                                          *}
{* Exit Conditions : If there is at least one entry available,*}
{* then the function will return a TRUE value.  Otherwise, the*}
{* result will be FALSE.                                         *}
{*****************************************************************}
function newuserok : boolean;
   begin
      if roomf(user_file)>0
      then newuserok:=true
      else newuserok:=false;
   end;
{*****************************************************************}
{* Function : Add_User                                           *}
{* Parameters : Name, Password                                   *}
{* Global Variables : Next_User                                  *}
{* Result : User_ID                                              *}
{* Entry Conditions : Name and Password are to be added to the*}
{* List of Users.                                                *}
{* Process : The Next_User id will be assigned to this Name   *}
{* Password pair and will be inserted into the List of Users. *}
{* Next_User will be incremented.                                *}
{* Exit Conditions : The function result will be the id       *}
{* assigned to the user.                                         *}
{*****************************************************************}
function add_user(name,password : char_string) : integer;
   var user_info : user_entry; { File entry buffer }
               r : integer;    { The physical record number of
                                 the new entry - not used here }
   begin  { add user }
      resetf(user_file);
   { Set up the Buffer }
      user_info.name:=name;
```

```
      user_info.password:=password;
   { Use the Next_User id and update it }
      user_info.id:=next_user; next_user:=next_user+1;
      add_user:=user_info.id;
      insertf(user_file,r,user_info);
   end; { add user }

modend.
```

Projects Module

```
{*****************************************************************}
{* Projects Module - Maintains the List of Projects for the    *}
{*                   Command Processor                           *}
{*****************************************************************}
module proj_list;
const max_str = 30;    { Maximum length of a string }
      max_item = 20;   { Maximum length of a menu item }
     { See File Access Module for description of Files Array }
      projects = 4;    { Slot number in Files Array for Projects}
      g_dir = 7;       { Slot number of Global Directory }

type char_string = packed array[1..max_str] of char;
     menu_string = packed array[1..max_item] of char;
       whichfile = 1..10;
           links = record
                     next,
                     prev : integer;
                   end;
{ List of Projects Type Definitions }
   state = (select_board, select_component, sel_done,
            connections, conn_done, placement, place_done,
            routing, route_done);
   proj_entry = record
                  link       : links;
                  id         : integer;     { Project ID number }
                  name       : char_string; { Project Name }
                  desc       : char_string; { Project Description }
                  completion : state;       { State of Completion }
                  user       : integer;     { Owner ID number }
                end;

{ Global Variable - Initialized by Read_Args routine in
  Argument Module }
var next_proj : external integer;  { Next project id to be assigned. }

{ File Access Declarations }
{ See File Access Module for details on the following }
external procedure readf(file_num : whichfile; var buffer : proj_entry;
                         var eolf : boolean);
external procedure writef(file_num : whichfile; var buffer : proj_entry;
                          var writeok : boolean);
external procedure insertf(file_num : whichfile; var rec_num : integer;
                           var buffer : proj_entry);
external procedure deletef(file_num : whichfile);
external procedure resetf(file_num : whichfile);
external function roomf(file_num : whichfile) : integer;

{ List building declarations }
{ See Menu Module for details on the following }
external procedure init_list;
external procedure build_list(item_text:menu_string;item_code:integer);
external function select_list : integer;
```

Projects Module

```
{ Returns number of Files Per Project - See Global Directory }
external function fpp : integer;

{*****************************************************************}
{* Function : New_P_OK - Is there room for another project?      *}
{* Exit Conditions : If there is room in both the Global         *}
{* Directory and the List of Projects, then New_P_OK will be     *}
{* TRUE.  Otherwise, FALSE will be returned.                     *}
{*****************************************************************}
function new_p_ok : boolean;
   begin
      if ((roomf(projects)>=1) and (roomf(g_dir)>=fpp))
      then new_p_ok:=true
      else new_p_ok:=false;
   end;

{*****************************************************************}
{* Procedure : Update_Proj_List - Update List of Projects        *}
{* Parameters : ID, Completion                                   *}
{* Entry Conditions : ID identifies the project in the List      *}
{* of Projects to be updated, and Completion is the new value    *}
{* for the state of the project.                                 *}
{* Process : The List of Projects will be scanned for project    *}
{* ID.  If found, its state of completion will be updated to     *}
{* Completion.  Nothing is done if ID is not found.              *}
{*****************************************************************}
procedure update_proj_list(id : integer; completion : state);
   var updated, eolf, writeok : boolean;
       proj_data : proj_entry;
   begin
      resetf(projects); updated:=false;
      repeat
         readf(projects,proj_data,eolf);
         if (proj_data.id=id)
            then begin  { We found it! }
                    proj_data.completion:=completion;
                    writef(projects,proj_data,writeok);
                    updated:=true;
                 end;
      until eolf or updated;
   end;

{*****************************************************************}
{* Function : Select_Project                                     *}
{* Parameters : User                                             *}
{* Result : One of the user's project IDs                        *}
{* Entry Conditions : User identifies whose projects to look     *}
{* up in the List of Projects.                                   *}
{* Process : All of User's projects are looked up in the List    *}
{* of Projects and the following information is inserted into    *}
{* a list - project Name and ID number.  The Names will be       *}
```

Projects Module

```
{* displayed in a menu and the user will be asked to select   *}
{* one.                                                       *}
{* Exit Conditions : The ID number corresponding to the user's*}
{* selection will be the function result.  If the User has no *}
{* projects in the List of Projects, then a value of Zero will*}
{* be returned.                                               *}
{************************************************************}
function select_project(user : integer) : integer;
    var item_text : menu_string;
                c : 1..max_item;
        proj_data : proj_entry;
     gotone, eolf : boolean;
    begin
       gotone:=false;
       resetf(projects); init_list;
       repeat
          readf(projects,proj_data,eolf);
          if ((proj_data.user=user) and (not eolf))
          then begin
                  gotone:=true;
                  for c:=1 to max_item do
                     item_text[c]:=proj_data.name[c];
                  build_list(item_text,proj_data.id);
               end;
       until eolf;
       if gotone
       then select_project:=select_list
       else select_project:=0;
    end;

{************************************************************}
{* Procedure : New_Project - Add a new project to the List    *}
{* Parameters : Name, Desc, User, ID                          *}
{* Global Variables : Next_Proj                               *}
{* Entry Conditions : Name, Desc(ription), and User define the*}
{* new project to be added to the List of Projects, and Next_ *}
{* Proj is the ID to be assigned to this project.             *}
{* Exit Conditions : ID is the Project ID assigned to the     *}
{* project.                                                   *}
{************************************************************}
procedure new_project(var name, desc : char_string;
                      user : integer; var id : integer);
   var proj_data : proj_entry;
       r : integer;
   begin
      resetf(projects);
     { Load the Buffer }
      proj_data.name:=name;
      proj_data.desc:=desc;
      proj_data.user:=user;
      proj_data.completion:=select_board;
     { Use the next sequential ID and update it }
```

```
      proj_data.id:=next_proj; next_proj:=next_proj+1;
     { Return the new ID the the Caller }
      id:=proj_data.id;
      insertf(projects,r,proj_data);
   end; { new project }

{*****************************************************************}
{* Procedure : Free_Project - Remove a project from the List     *}
{* Parameters : Project_ID                                       *}
{* Entry Conditions : Project_ID identifies the project to be    *}
{* removed from the List of Projects.                            *}
{* Process : List of Projects will be scanned for Project_ID.    *}
{* If it is found, it will be deleted.  Nothing happens if the   *}
{* project isn't in the list.                                    *}
{*****************************************************************}
procedure free_project(project_id : integer);
   var eolf : boolean;
       proj_data : proj_entry;
   begin
      if project_id<>0
      then begin
             resetf(projects);
             repeat
                readf(projects,proj_data,eolf);
                if proj_data.id=project_id
                then deletef(projects);
             until (eolf or (proj_data.id=project_id));
           end;
   end;

{*****************************************************************}
{* Function : Get_State - What is the state of the Project?      *}
{* Parameters : ID                                               *}
{* Result : The current State of Completion of the project       *}
{* Entry Conditions : ID identifies which project to look up     *}
{*****************************************************************}
function get_state(id : integer) : state;
   var eolf : boolean;
       proj_data : proj_entry;
   begin
      resetf(projects);
      repeat
         readf(projects,proj_data,eolf);
         if (proj_data.id=id)
         then get_state:=proj_data.completion;
      until (eolf or (proj_data.id=id));
   end;

{*****************************************************************}
{* Procedure : Get_Name - What is the name of the project?       *}
{* Parameters : ID, Name                                         *}
{* Entry Conditions : ID identifies the project to look up       *}
```

Projects Module

```
(* Exit Conditions : Name will contain the name of the project*)
(* if ID exists.  If ID isn't in the List of Projects, name   *)
(* will be set to all spaces.                                 *)
(************************************************************)
procedure get_name(id : integer; var name : char_string);
   var eolf : boolean;
       proj_data : proj_entry;
   begin
      name:='                                    ';
      resetf(projects);
      repeat
         readf(projects,proj_data,eolf);
         if ((proj_data.id=id) and (not eolf))
         then name:=proj_data.name;
      until (eolf or (proj_data.id=id));
   end;

modend.
```

Argument Module

```
{*****************************************************************}
{* Argument Module - Responsible for passing arguments between*)
{* the Command Processor and the other layout modules.        *}
{*****************************************************************}
module arguments;
const args = 5;
     g_dir = 7;
type
        wf = 1..10;
{ Argument File Definitions }
     state = (select_board, select_component, sel_done,
             connections, conn_done, placement, place_done,
             routing, route_done);
   modules = (cp, selecter, connecter, placer, router, os);
{ Arg_Header contains information the CP needs to know. }
arg_header = record
                project_id,          { Current Project }
                user_id,             { Current User }
                error_code : integer;{ Current Error }
                completion : state;  { State of Completion }
                last_mod   : modules;{ Last module executed }
             end;
{ Saved_State keeps track of the next IDs to assign }
saved_state = record
                next_id,
                next_user,
                next_proj : integer;
                proj_info : arg_header;
              end;

  filespec = array[1..12] of char;
  drive_id = (A, B);
      link = record
                next,
                prev : integer;
             end;
     where = record
               linked    : boolean;
               file_name : filespec;
               drive     : drive_id;
               disk_id,
               recs_avail,
               rec_len,
               first,
               free      : integer;
             end;
   arg_entry = record
                 links : link;
                 case boolean of
                 false:(header : saved_state);
                  true:(file_entry : record
                                       file_num : wf;
```

Argument Module

```
                                        file_loc : where;
                                       end);
            end;

{ Global Variables }
var next_proj, next_user, next_id : integer;

{ See the LinkFile Module for details on the following }
external procedure readf(f : wf; var buffer:arg_entry; var eolf:boolean)
;
external procedure writef(f : wf; var buffer:arg_entry; var writeok:bool
ean);
external procedure resetf(f : wf);
external procedure insertf(f:wf; var rec_num:integer; var buffer : arg_e
ntry);
external procedure deletef(f : wf);

{ See the Global Directory Module for details on the following }
external procedure update_file(m : modules; project : integer;
                               file_num : wf; file_loc : where);
external function get_loc(m : modules; project : integer;
                          var file_num : wf; var file_loc : where) : boo
lean;

{*****************************************************************}
{* Procedure : Read_Args - Read in arguments passed              *}
{* Parameters : Info                                             *}
{* Entry Conditions : The Argument File contains the current     *}
{* state of the system.                                          *}
{* Process : The current state is sent back to the Command       *}
{* Processor in Info.  If the last module executed updated       *}
{* any files, then the Global Directory has to be updated.       *}
{*****************************************************************}
procedure read_args(var info : arg_header);
   var eolf : boolean;
       buffer : arg_entry;
   begin
      resetf(args);
      readf(args,buffer,eolf);
      info:=buffer.header.proj_info;;
      next_proj:=buffer.header.next_proj;
      next_user:=buffer.header.next_user;
      next_id:=buffer.header.next_id;
      repeat  { Read current state of files }
         readf(args,buffer,eolf);
         if not eolf
         then begin
                deletef(args);
                with buffer.file_entry do
              { Update the Global Directory to match }
                 update_file(info.last_mod,info.project_id,
                 file_num,file_loc);
```

```
            end;
      until eolf;
   end;

{*****************************************************************}
{* Procedure : Update_Header - Save current state                *}
{* Parameters : Info                                             *}
{* Entry Conditions : Info is the current state of the system    *}
{* to be save in the Argument File.                              *}
{*****************************************************************}
procedure update_header(info : arg_header);
   var eolf, writeok : boolean;
       buffer : arg_entry;
   begin
      resetf(args);
      readf(args,buffer,eolf);
      buffer.header.next_proj:=next_proj;
      buffer.header.next_user:=next_user;
      buffer.header.next_id:=next_id;
      buffer.header.proj_info:=info;
      writef(args,buffer,writeok);
   end;

{*****************************************************************}
{* Procedure : Load_Args - Load Argument File with files         *}
{* Parameters : M, Project                                       *}
{* Entry Conditions : Module and Project identify which files    *}
{* are to be loaded.                                             *}
{* Process : The Global Directory is scanned for files with      *}
{* matching Module and Project parameters.  When found, their    *}
{* locations are inserted into the Argument File.                *}
{*****************************************************************}
procedure load_args(m : modules; project : integer);
   var buffer : arg_entry;
            r : integer;
         eolf : boolean;
   begin
     { Reset the Directory and the Argument files, and skip the
       Argument File Header record. }
      resetf(g_dir); resetf(args); readf(args,buffer,eolf);
      repeat { Scan the Global Directory }
        with buffer.file_entry do
           eolf:=get_loc(m,project,file_num,file_loc);
        if not eolf
        then insertf(args,r,buffer);
      until eolf;
   end;

modend.
```

DiskList Module

```
{*****************************************************************}
{* DiskList Module - Keeps track of allocated and available    *}
{*                   Diskette space.                             *}
{* A record is maintained for each diskette in the system.     *}
{* Each diskette may be either assigned to a specific project, *}
{* or assigned to a user.  If assigned to a user, then it is   *}
{* avaialable for assignment to a project created by that user *}
{* only.                                                       *}
{*****************************************************************}
module disk_list;
const max_space = 376; { Maximum space available in Kbytes }
      disks = 6;
type
    wf          = 1..10;
    links       = record
                      next,
                      prev : integer;
                  end;
    disk_assgn = record
                      link : links;
                  disk_id,            { Which disk is it }
              free_space : integer;   { How much room is available }
           case assigned : boolean of
      true : (project_id : integer);
        false : (user_id : integer);
                  end;

{ Global Variable initialized by Read_Args - See Argument Module for det
ails }
var next_id : external integer;

{ File Access Declarations }
{ See the LinkFile Module for details on the following }
external procedure readf(f:wf; var buffer:disk_assgn; var eolf:boolean);
external procedure writef(f:wf; var buffer:disk_assgn; var writeok:boole
an);
external procedure insertf(f:wf; var rec_num:integer;var buffer:disk_ass
gn);
external procedure resetf(f:wf);

{ See the DiskID Module for details on the following }
external procedure new_disk(disk_id : integer) ;

{*****************************************************************}
{* Function : Get_Disk_ID - Assign a project file to a disk    *}
{* Parameters : User, Project, Size                            *}
{* Result : Disk_ID assigned to file                           *}
{* Entry Conditions : User identifies the owner of the file,   *}
{* Project identifies the project to which the file belongs,   *}
{* and Size is the number of KBytes required by the file.      *}
{* Process : The Disks File is scanned for a diskette already  *}
{* assigned to Project with enough free space on it.  If one   *}
```

DiskList Module

```
(* can't be found, then the file is scanned for a diskette    *)
(* belonging to User.  If one is found, it is assigned to pro-*)
(* ject.  If the User has no free disks, then a new diskette  *)
(* will be added to the system and assigned to the project.   *)
{*****************************************************************}
function get_disk_id(user, project, size : integer) : integer;
   var done, eolf, writeok : boolean;
       disk_data : disk_assgn;
       r : integer;
   begin
      resetf(disks); done:=false;
      repeat
        { Look for a diskette already assigned to Project with
          enough free space on it for the file }
        readf(disks,disk_data,eolf);
        if ((disk_data.assigned) and
            (disk_data.project_id=project) and
            (disk_data.free_space>=size))
           then with disk_data do
                begin
                  { Found One! }
                   get_disk_id:=disk_id;
                   free_space:=free_space-size;
                   writef(disks,disk_data,writeok);
                          done:=true;
                end
 { Next, look for a free disk assigned to User }
    else if ((not disk_data.assigned) and
             (disk_data.user_id=user))
           then with disk_data do
                begin
                { Assign it to Project }
                   get_disk_id:=disk_id;
                   free_space:=max_space-size;
                   assigned:=true;
                   project_id:=project;
                   writef(disks,disk_data,writeok);
                   done:=true;
                end;
      until (eolf or done);
      if not done
      then with disk_data do
           begin
             { Have to make a new diskette available }
              disk_id:=next_id; next_id:=next_id+1;
              get_disk_id:=disk_id;
              free_space:=max_space;
              assigned:=true;
              project_id:=project;
              insertf(disks,r,disk_data);
             { Label and identify the new diskette }
               new_disk(disk_id);
```

```
            end;
   end;

{*****************************************************************}
{* Procedure : Free_Disk - Deallocate diskette space             *}
{* Parameters : User, Project                                    *}
{* Entry Conditions : Project identifies the project which is    *}
{* is no longer active, and User identifies who the released     *}
{* space will be assigned to.                                    *}
{*****************************************************************}
procedure free_disk(user,project : integer);
   var eolf, writeok : boolean;
       disk_data : disk_assgn;
   begin
      resetf(disks);
      repeat
         readf(disks,disk_data,eolf);
         if ((disk_data.assigned) and
             (disk_data.project_id=project))
         then begin
                 disk_data.assigned:=false;
                 disk_data.user_id:=user;
                 disk_data.free_space:=max_space;
                 writef(disks,disk_data,writeok);
              end;
      until eolf;
   end;

modend.
```

Global Directory Module

```
{*****************************************************************}
{* Global Directory Module - Maintains the directory of all    *}
{* files that the system has knowledge of.                     *}
{*****************************************************************}
{* There are three types of files maintained in the directory.*}
{* The majority of the entries are for actual files associated*}
{* with either projects or layout modules.  These entries      *}
{* contain all the information necessary to locate and use the*}
{* file.  The second type of entry is the "Template Entry."    *}
{* A template entry is identified by a Disk_ID field with a    *}
{* value of zero.  Whenever a new project is created, a new    *}
{* file is created to match each template entry.               *}
{* The third type of file entry identifies executable files.   *}
{* These entries have the OS (Operating System) flag set to    *}
{* identify them.  There will be one entry corresponding to    *}
{* each separate module of the layout system.                  *}
{*****************************************************************}
module global_directory;
const g_dir = 7;
      dump = 8; { Temporary slot used for files }
type
 whichfile = 1..10;
{ Global Directory Type Definitions }
  { Which Module(s) can access the file? }
   modules = (cp,selecter,connecter,placer,router,os);
  { System Dependent File Specification }
  filespec = packed array[1..12] of char;
  { System Dependent Drive Identification }
  drive_id = (A, B);
  { Where is the file located ?}
     where = record
               linked    : boolean;   { Linked or sequential access }
               file_name : filespec;  { Name of the File }
                   drive : drive_id;  { Drive in which to
                                        Mount Diskette }
                 disk_id,             { Which diskette the file is on }
                 rec_len,             { The record length (power of two)
}
              recs_avail,             { Number of unused records in file
;
                   first,             { Physical record number of first
                                        logical record }
                    free : integer;   { Physical record number of
                                        free space list }
             end;
      link = record
               next,
               prev : integer;
             end;
  gd_entry = record
               links     : link;
               module_id : packed array[modules] of boolean;
```

```
                                { True value means the module has
                                  access to the file }
                    file_num : whichfile; { Files Array Index }
                  project_id,  { Project to which file belongs. A
                                  value of Zero indicates a file
                                  that is used for all projects }
                    how_many : integer; { How Many records have
                                          been allocated }
                    file_loc : where;
                  end;

{ See the File Access Module for details on the following }
external procedure statef(file_num:whichfile; var first,free : integer);
external function roomf(file_num : whichfile) : integer;
external procedure resetf(file_num : whichfile);
external procedure initf(file_num : whichfile; file_loc : where);
external procedure closef(file_num : whichfile);
external procedure close_all(drive : drive_id);
external procedure readf(file_num : whichfile; var buffer : gd_entry;
                         var eolf : boolean);
external procedure writef(file_num : whichfile; var buffer : gd_entry;
                          var writeok : boolean);
external procedure insertf(file_num : whichfile; var rec_num : integer;
                           var buffer : gd_entry);
external procedure deletef(file_num : whichfile);
external procedure createf(file_num : whichfile; how_many : integer);
external procedure erasef(file_num : whichfile);
external procedure run_file(file_name : filespec; drive : drive_id;
                            disk_id : integer);
external procedure init_files;

{ See the Menu Module for details on the following }
external procedure init_menu;

{ See the TermIO Module for details on the following }
external procedure init_term;

{ See the Disk List Module for details on the following }
external function get_disk_id(user, project, space : integer):integer;

{*****************************************************************}
{* Procedure : New_files                                         *}
{* Parameters : User, Project                                    *}
{* Entry Conditions : User and Project identify a new project    *}
{* that the user is creating.                                    *}
{* Process : The Global Directory is scanned for Template        *}
{* Entries.  A Template Entry is identified by a value of Zero   *}
{* in the Disk_ID field.  For each Template Entry found, a new   *}
{* file is created and an entry is made in the Global            *}
{* Directory.                                                    *}
{*****************************************************************}
procedure new_files(user, project : integer);
```

Global Directory Module

```
   var new_gde : gd_entry;   { Buffer for New Entries }
             r : integer;    { R is the physical record number of
                               newly inserted entries - not used here. }
     maxkbytes : integer;    { MaxKbytes is the number of Kbytes required
 by the
                               file. }
      maxbytes : real;       { The number of bytes required by the file }
          eolf : boolean;    { End of Directory indicator }
   begin
      resetf(g_dir);
      repeat
         readf(g_dir,new_gde,eolf);
         if new_gde.file_loc.disk_id=0
         then begin  { Located a Template File }
                 with new_gde.file_loc do
                 begin
                    maxbytes:=rec_len * new_gde.how_many;
                    maxkbytes:=round(maxbytes / 1024) + 1;
                 { Allocate Diskette space for the file }
                    disk_id:=get_disk_id(user,project,maxkbytes);
                 end;
                { Assign the file to the Project }
                 new_gde.project_id:=project;
                { Create the new file }
                 initf(dump,new_gde.file_loc);
                 createf(dump,new_gde.how_many);
                 closef(dump);
                { Add it to the Directory }
                 insertf(g_dir,r,new_gde);
              end;
      until eolf;
   end;

{*****************************************************************}
{* Function : FPP - Files Per Project                            *}
{* Result : The number of files created for every project        *}
{*****************************************************************}
function fpp : integer;
   begin
      fpp:=3 { Value depends on current system configuration }
   end;

{*****************************************************************}
{* Procedure : Kill_GDE                                          *}
{* Parameters : Project_ID                                       *}
{* Entry Conditions : Project_ID identifies the project whose    *}
{* files are to be removed from the Directory.                   *}
{* Process : The Directory is scanned looking for Entries with   *}
{* a Project_ID that matches the parameter. When found, those    *}
{* Directory Entries are deleted and the corresponding file is   *}
{* Purged from the system.                                       *}
{*****************************************************************}
```

Global Directory Module

```
procedure kill_gde(project_id : integer);
   var gde : gd_entry; { Directory Entry Buffer }
      eolf : boolean;  { End of Directory flag }
   begin { kill }
      if project_id<>0
         then begin
                 resetf(g_dir);
                 repeat
                    readf(g_dir,gde,eolf);
                    if gde.project_id=project_id
                    then begin  { Found one! }
                            deletef(g_dir);
                            initf(dump,gde.file_loc);
                            erasef(dump);
                         end;
                 until eolf;
              end;
   end; { kill }

{*************************************************************}
{* Procedure : UpdateGD   Update Global Directory            *}
{* Process : Before the Command Processor Finishes execution, *}
{* the new status of all of the files must be recorded in the *}
{* Directory to keep it current.  As records are added to or  *}
{* deleted from the Command Processor Files, the First, Free, *}
{* and Recs_Avail parameters are subject to change.           *}
{*************************************************************}
procedure updategd;  { Must be called before CP terminates!!!}
   var eolf, writeok : boolean;
                gde : gd_entry;
first, free, recs_avail : integer; { Parameters to be updated }
   begin
      resetf(g_dir);
      repeat
         readf(g_dir,gde,eolf);
         if ((not eolf) and gde.module_id[cp])
            then begin
                   { Get the current state of the file }
                    statef(gde.file_num,first,free);
                    gde.file_loc.first:=first;
                    gde.file_loc.free:=free;
                   { Get the number of available records }
                    gde.file_loc.recs_avail:=roomf(gde.file_num);
                    writef(g_dir,gde,writeok);
                 end;
      until eolf;
     { Make sure all the files get updated in the OS directory }
      close_all(A);
      close_all(B);
   end;

{*************************************************************}
```

Global Directory Module

```
{* Procedure : Update_File - Update a file's  GD Entry        *}
{* Parameters : File_Num, Project, File_Loc                    *}
{* Entry Conditions : File_Num and Project identifies the file*}
{* whose Global Directory Entry must be updated.              *}
{* Process : The file is located in the Directory and the     *}
{* entry is updated with the information in File_Loc.         *}
{*************************************************************}
procedure update_file(m : modules; project : integer;
                      file_num : whichfile;  file_loc : where);
   var eolf, writeok : boolean;
       gde : gd_entry;
   begin
      resetf(g_dir); writeok:=false;
      repeat
         readf(g_dir,gde,eolf);
         if ((gde.file_num=file_num) and
             (gde.project_id=project) and
             (gde.module_id[m]))
         then begin  { This is the Entry to be Updated }
                 gde.file_loc:=file_loc;
                 writef(g_dir,gde,writeok);
              end;
      until eolf or writeok;
   end;

{*************************************************************}
{* Function : Get_Loc - Locate files for other Modules        *}
{* Parameters : Module, Project, File_Num, File_Loc           *}
{* Result : EOLF condition                                    *}
{* Entry Conditions : Module and Project identify which files *}
{* are to be looked for in the Directory.                     *}
{* Process : The Directory is scanned from the current        *}
{* looking for files that belong to the Module and Project.   *}
{* If one is found, its number and location are returned to   *}
{* the caller.  This function should be called repeatedly     *}
{* until an End of File condition is signalled.               *}
{* Exit Conditions : File_Num and File_Loc are returned, and  *}
{* the function returns a FALSE value, if a file was located  *}
{* that belonged to the module and project.  If the End of    *}
{* file is reached, then the function returns a TRUE value.   *}
{*************************************************************}
function get_loc(m : modules; project : integer;
    var file_num : whichfile; var file_loc : where) : boolean;
   var eolf : boolean;
       gde : gd_entry;
   begin
      repeat
        readf(g_dir,gde,eolf);
      until (eolf or
                ((gde.module_id[m]) and
                 ((gde.project_id=project) or (gde.project_id=0))));
      file_num:=gde.file_num;
```

```
      file_loc:=gde.file_loc;
      get_loc:=eolf;
   end;

{*****************************************************************}
{* Procedure : Init_All  Initialize Everything!                  *}
{* Process : After some Miscellaneous initialization routines *}
{* are called, the Global Directory is Initialized.  Then all *}
{* of the Command Processor Files are looked up in the        *}
{* Directory, and they are initialized.  No access may be made*}
{* to a file until it is initialized!                         *}
{*****************************************************************}
procedure init_all;
   var file_loc : where;
          eolf : boolean;
           gde : gd_entry;
   begin
      init_files; { Set up the file array with all files closed}
      init_menu;  { Set the menu pointers to nil }
      init_term;  { Perform all necessary terminal setup, if any }
      { The global directory must be initialized }
      with file_loc do
      begin
        { You have to know where to find the Directory }
         disk_id   := 1;
         file_name := 'GLOBAL.DIR  ';
         drive     := A;
         rec_len   := sizeof(gd_entry);
         linked    := true;
         first     := 0;  { This should'nt change!!!}
      end;
      initf(g_dir,file_loc);
     { Free and Recs_Avail parameters still not set }
      resetf(g_dir);
     { Look up Directory entry in the Directory to set them }
      repeat
         readf(g_dir,gde,eolf);
      until gde.file_num=g_dir;
      closef(g_dir); initf(g_dir,gde.file_loc);
      { Now the rest of the files can be located }
      resetf(g_dir);
      repeat
        readf(g_dir,gde,eolf);
        if ((not eolf) and
            (gde.file_num<>g_dir) and
            (gde.module_id[cpl) and
            (gde.project_id=0))
          then initf(gde.file_num,gde.file_loc);
      until eolf;
   end;

{*****************************************************************}
```

Global Directory Module

```
{* Procedure : Exec - Execute an Operating System Program      *}
{* Parameters : Module                                         *}
{* Entry Conditions : Module identifies the Program to be      *}
{* executed.  The Command Processor will no longer be in       *}
{* control.                                                    *}
{************************************************************}
procedure exec(m : modules);
   var gde : gd_entry;
       eolf : boolean;
   begin
     { To execute OS, all that is necessary is to return from
       this procedure, after which the program will end! }
      if m<>os
     { Otherwise, the proper module must be executed. }
      then begin
              resetf(g_dir);
              repeat
                 readf(g_dir,gde,eolf);
                 if (gde.module_id[m] and gde.module_id[os])
                 then with gde.file_loc do
                      run_file(file_name,drive,disk_id);
              until eolf;
           end;
   end; { If you get to here, then the OS will take control. }

modend.
```

Command Processor Module

```
{*****************************************************************}
{*  Command Processor Module                                     *}
{*****************************************************************}
{*   The following files must exist on the master CP disk        *}
{*   (Disk number 1):                                            *}
{*   GLOBAL.DIR - describes the parameters of the rest           *}
{*                of the files                                   *}
{*   MENUS.CP   - Contains menus 1 and 2                         *}
{*                (See Select_Option and Get_Next_Module         *}
{*                for menu contents)                             *}
{*   HELP.CP    - Contains all of the system Help                *}
{*                messages.                                      *}
{*   USERS.CP   - A list of all Users and their ID's             *}
{*   PROJECT.CP - A list of all Projects and their ID's          *}
{*   DISKS.CP   - Contains Diskette space allocation             *}
{*   ARGS.CP    - The argument file is the means of              *}
{*                communication between the system               *}
{*                modules.  It also contains "Current            *}
{*                State" information that must be saved          *}
{*                between Command Processor sessions.            *}
{*                In particular, the Command_Processor           *}
{*                Global Variables are initialized with          *}
{*                the following information:                     *}
{*   project_id - Number of Last project accessed                *}
{*   user_id    - Identity of current user                       *}
{*   error_code - Error conditions of other modules              *}
{*   completion - State of completion of project_id              *}
{*   Exit Conditions : The Global Directory and Argument         *}
{*   Files are updated, and either the program is exited, or     *}
{*   the next layout module is invoked.                          *}
{*****************************************************************}
program command_processor;
const max_prompt = 30;      { Maximum length of a prompt }
        max_num = 10;       { Maximum length of a number }
        max_str = 30;       { Maximum length of a string }

type
{ Query Type Definitions }
             unit = (mils, inches, mm, scalar);
    prompt_string = packed array[1..max_prompt] of char;
            p_len = 1..max_prompt;
         position = 1..max_str;
      string_case = (upper, lower, none);
      char_string = packed array[1..max_str] of char;
            yesno = (yes, no, i_dunno);
            boxes = (q, m, h);

{ Global Directory Type Definitions }
   modules = (cp,selecter,connecter,placer,router,os,undefined);
             { A list of the possible values for the next module
               to be executed }
```

Command Processor Module

```
{ List of Projects Type Definitions }
   state = (select_board, select_component, sel_done,
            connections, conn_done, placement, place_done,
            routing, route_done);
           { A list of the possible values for the state of
             completion of a project }

{ Argument File Type Definitions }
{ Arg_Header defines the information the Command_Processor reads
  from the ARGS.CP file to initialize the global variables.}
  arg_header = record
                  project_id,
                  user_id,
                  error_code : integer;
                  completion : state;
                  next_mod   : modules;
               end;

var { Global Variables }
     current : arg_header;
       valid : boolean;    { Indicates the validity of a user }

{ Query Declarations }
{ See the Query Modules for an explanation of these routines }
external procedure querystr(prompt        : prompt_string;
                            prompt_length : p_len;
                            min, max      : position;
                        var answer        : char_string;
                        var string_length : position;
                            make_case     : string_case;
                            help_index    : integer);
external procedure query_yn(prompt        : prompt_string;
                            prompt_length : p_len;
                        var answer        : yesno;
                            help_index    : integer);

{ Screen I.O Declarations }
{ See the TermIO Module for an explanation of these routines. }
external procedure goto_box(box : boxes; x,y : integer);
external procedure clear_line(box : boxes; y : integer);
external procedure clear_box(box : boxes);
external function get_count(box : boxes) : integer;
external procedure input_error(err_msg : char_string);

{ See the Global Directory Module for details on the following }
external procedure init_all;
external procedure new_files(user, project : integer);
external procedure kill_gde(project : integer);
external procedure updategd;
external procedure exec(next_mod : modules);

{ See the Users Module for details on the following }
```

Command Processor Module

```
external function lookup(user_name : char_string;
                         var password : char_string;
                         var id : integer) : boolean;
external function add_user(user_name,password:char_string):integer;
external function newuserok : boolean;

{See the Projects Module for details on the following }
external procedure update_proj_list(id : integer; completion : state);
external function select_project(user_id : integer) : integer;
external function get_state(id : integer) : state;
external procedure get_name(id:integer;var name:char_string);
external function new_p_ok : boolean;
external procedure new_project(var name, desc : char_string;
                               user : integer; var id : integer);
external procedure free_proj(project_id : integer);

{ See the Disk List Module for details on the following }
external procedure free_disk(user, project : integer);

{See the Menu Module for details on the following function }
external function menu(num : integer) : integer;

{ See the Argument Module for details on the following }
external procedure read_args(var info : arg_header);
external procedure update_header(info : arg_header);
external procedure load_args(next_mod : modules; project : integer);

{*****************************************************************}
{* Procedure Name : Resolve                                      *}
{* Parameters : Error_Code                                       *}
{* Entry Conditions : Error_Code identifies a particular error*}
{* condition that another module was unable to handle.           *}
{* Exit Conditions : If possible, the error condition will be *}
{* corrected and the error_code will be reset to zero.           *}
{* In the current implementation, this is a dummy routine, but*}
{* it is provided for future expansion.                          *}
{*****************************************************************}
procedure resolve(var error_code : integer);
   begin
      write('***** ERROR ',error_code:3,' *****');
      error_code:=0
   end;

{*****************************************************************}
{* Procedure : Get_Identity                                      *}
{* Parameters : ID, Valid                                        *}
{* Entry Conditions : The identity of the user is unknown.       *}
{* Exit Conditions : ID will identify the user if he exists.     *}
{* Valid is a flag that will indicate if the user exists.        *}
{*****************************************************************}
procedure get_identity(var id:integer; var valid:boolean);
  const no_password =                                  :
```

Command Processor Module

```
   var      name,                    { The name of the user }
        password : char_string;      { His Password }
        name_len : position;         { The number of characters
                                       in the user's name }
           found : boolean;          { A Flag indicating whether
                                       or not the user already
                                       exists }
          answer : yesno;            { Will indicate if the user
                                       typed his name correctly }


{*****************************************************************}
{* Procedure : Create_User                                       *}
{* Parameters : Name                                             *}
{* Outer Level Variables : Password, ID, Valid                   *}
{* Entry Conditions : Name was not found in the List of          *}
{* Current Users.                                                *}
{* Exit Conditions : If there is room for another user in the    *}
{* List of Current Users and the operator indicated a desire     *}
{* to become a user, then ID and Password will be updated and    *}
{* Vaild will be set to true.  Otherwise, Valid will be false.   *}
{*****************************************************************}
   procedure create_user(name : char_string);
      var answer : yesno;          {Will indicate whether or not
                                    the operator wishes to become
                                    a user }

{*****************************************************************}
{* Procedure : Get_Password                                      *}
{* Parameters : Password                                         *}
{* Entry Conditions : A new user must be assigned a Password     *}
{* Exit Conditions : If the user indicated that he didn't want   *}
{* a password, then Password will be set to all spaces.          *}
{* Otherwise, Password will be set to whatever character         *}
{* string he entered.                                            *}
{*****************************************************************}
      procedure get_password(var password : char_string);
         var answer : yesno;       { Will indicate whether or not
                                     a password is desired }
                len: position;     { The length of the password }
         begin
            query_yn('Do you want a Password?        ',23,
                     answer,4);
            if answer=yes then
               querystr('What will your Password be?    ',27,
                        5,20,password,len,none,5)
               else password:=no_password;
         end;  { get_password }

      begin { create_user }
         if newuserok      { There IS room for another one }
         then begin
```

```
                query_yn('Do you wish to become a user? ',29,
                         answer,3);
                if answer=yes
                   then begin { A New User!!!}
                           get_password(password);
                           id:=add_user(name,password);
                           valid:=true;
                        end
                   else valid:=false;
             end
          else input_error('No More Room for New Users!    ');
       end; { create_user }

{*****************************************************************}
{* Function : Verify_Identity                                    *}
{* Parameters : Password                                         *}
{* Entry Conditions : The users name was located in the List     *}
{* of Current Users, and Password contains the password that     *}
{* the user originally specified.                                *}
{* Exit Conditions : If the comparison string entered by the     *}
{* user matches his original Password, or if the original is     *}
{* all spaces, then the function returns a True result.          *}
{* Otherwise, a False value is returned.  The user get three     *}
{* tries to get the password right.                              *}
{*****************************************************************}
   function verify_identity(password : char_string) : boolean;
      var testword : char_string;  { The string to compare with
                                     the original Password }
                len : position;     { The number of characters in
                                     testword }
              tries : integer;      { Counts the number of
                                     attempts to match }
              valid : boolean;      { Indicates result of com-
                                     parison }
      begin
         if password=no_password
            then verify_identity:=true
            else begin
                    tries:=0; valid:=false;
                    repeat
                       querystr('Prove It!                          ',9,
                              5,20,testword,len,none,6);
                       if testword=password then valid:=true;
                       tries:=tries+1;
                    until (valid or (tries=3));
                    verify_identity:=valid;
                 end
      end; { verify_identity }

   begin { get_identity }
     repeat
        querystr('Who Are You ?                    ',12,
```

```
                    2,20,name,name_len,upper,1);
          found:=lookup(name,password,id); answer:=yes;
          if not found
             then begin
                    clear_line(q,-1);
                    write(name);
                    query_yn('Is your name correct?          ',21,
                             answer,2);
                    clear_line(q,-1);
                  end;
       until answer=yes;
       if not found
          then create_user(name)
          else valid:=verify_identity(password);
    end; { get_identity }

{*****************************************************************}
{* Procedure : Select_Option                                     *}
{* Parameters : User_ID, Project_ID, Completion, Next_Module     *}
{* Entry Conditions : User_ID identifies the current user.       *}
{* Exit Conditions : Project_ID, Completion, and Next_Module     *}
{* will have been updated to reflect the current status of the*}
{* project selected by the user.                                 *}
{*****************************************************************}
procedure select_option(var user_id : integer;
                        var project_id : integer;
                        var completion : state;
                        var next_module : modules);

    var  name : char_string;    { Will contain the name of the
                                  current project }
         selection : integer;   { The Menu Selection Index }


{*****************************************************************}
{* Function : Get_Next_Module                                    *}
{* Parameters : Completion                                       *}
{* Entry Conditions : The user has selected a project and        *}
{* desires to work on it.  Completion identifies the current     *}
{* state of the selected project.                                *}
{* Exit Conditions : The function result is the Next Step in     *}
{* Layout Process that the user has selected.  A given step      *}
{* may be repeated as often as desired, but may only be per-     *}
{* formed if all of the previous steps have been completed.      *}
{*****************************************************************}
    function get_next_module(completion : state) : modules;
       var selection : integer;       { Will contain the selection
                                        from Menu 2 }
           answer : yesno;            { Will contain response to
                                        query on repeating a step}
           highest, next: modules;    { Will identify the highest
                                        allowable step and the
```

```
                                        user's selection }
      begin { get_next_module }
         case completion of
          select_board, select_component : highest:=selecter;
                   sel_done, connections : highest:=connecter;
                    conn_done, placement : highest:=placer;
         place_done, routing, route_done : highest:=router;
         end;
         repeat
            repeat
               selection:=menu(2);
               { 1 : Selecter
                 2 : Connecter
                 3 : Placer
                 4 : Router }
               case selection of
               1 : next:=selecter;
               2 : next:=connecter;
               3 : next:=placer;
               4 : next:=router;
               end;
               if next>highest
               then input_error('You Can''t do that yet!        ');
            until next<=highest;
            if next<highest
               then query_yn('You want to redo this step?    ',27,
                             answer,16)
               else answer:=yes;
         until answer=yes;
         get_next_module:=next;
      end; { get_next_module }

{*****************************************************************}
{* Procedure : Create_Project                                    *}
{* Parameters : User, ID, Name                                   *}
{* Entry Conditions : The user has decided to create another     *}
{* project.  User is the identification number of the current    *}
{* user to whom the new project will be assigned.                *}
{* Exit Conditions : If there is room for another project,       *}
{* then ID will contain the project identification number that   *}
{* was assigned to it, and Name will contain the user assigned   *}
{* name. If there is not room for another project to be          *}
{* defined, then ID will be zero.                                *}
{*****************************************************************}
   procedure create_project(user : integer; var id : integer;
                            var name : char_string);
      var desc : char_string; { User's description of project }
          ans_len : position; { Number of characters in name }
      begin
         if new_p_ok
         then begin  { There IS room for another project }
              querystr('What do you want to call it?   ',28,1,20,
```

```
                    name,ans_len,none,20);
            querystr('Give a brief description :     ',26,1,30,
                    desc,ans_len,none,21);
          { Add the project to List of Projects }
            new_project(name,desc,user,id);
          { Allocate Diskette space for the project }
            new_files(user,id);
            end
        else begin
            input_error('No More Room for Projects!     ');
            id:=0;
            end;
    end;

{*****************************************************************}
{* Procedure : Kill_Project                                      *}
{* Parameters : User, Project                                    *}
{* Entry Conditions : The user has decided to destroy the        *}
{* currently selected project identified by Project.             *}
{* Exit Conditions : The project will have been removed from     *}
{* the List of Projects and the diskette space will have been    *}
{* de-allocated.  Also, Project will have been reset to zero     *}
{* to indicate that there is no longer a current project.        *}
{*****************************************************************}
  procedure kill_project(user:integer; var project:integer);
    begin
      { Remove the Project from the Directory }
       kill_gde(project);
      { Remove the project from the List of Projects }
       free_project(project);
      { De-allocate the Diskette space }
       free_disk(user,project);
      { Reset to No Current Project }
       project:=0;
    end;

  begin { Select Option }
    if project_id<>0
    then begin { There IS a currently selected project }
           { Make sure the List of Projects is Up to Date }
            update_proj_list(project_id,completion);
            get_name(project_id,name);
         end
    else name:='No Current Project Selected.  ';
    next_module:=undefined;
    repeat
      clear_line(q,3); write('Current Project :',name);
      selection:=menu(1);
      { 1 : Continue Project
        2 : Switch Projects
        3 : Create New Project
        4 : Discard Project
```

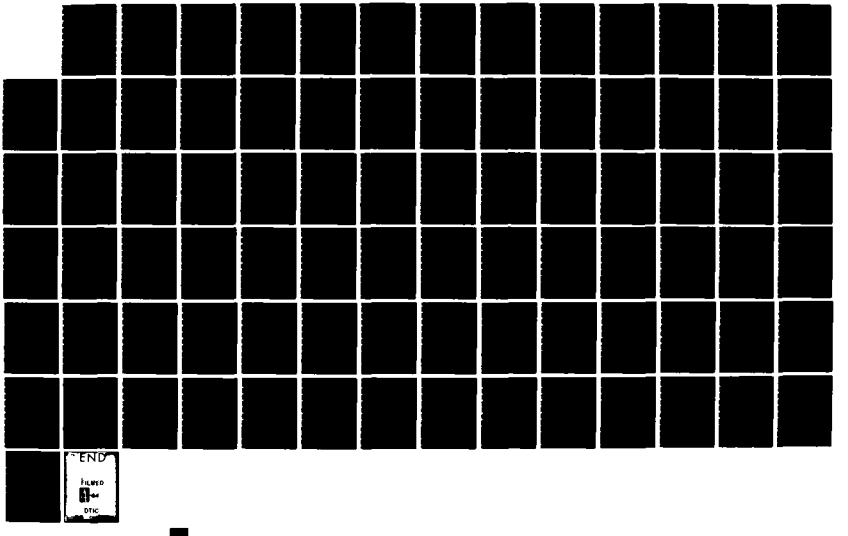
```
            5 : Exit to Operating System )
          case selection of
          1 : if project_id=0
              then input_error('No current project selected!  ')
              else next_module:=get_next_module(completion);
          2 : begin
               project_id:=select_project(user_id);
               if project_id=0
               then input_error('You have no Projects defined! ')
               else begin
                       completion:=get_state(project_id);
                       get_name(project_id,name);
                    end;
              end;
          3 : create_project(user_id, project_id, name);
          4 : if project_id<>0
              then begin
                      kill_project(user_id, project_id);
                      name:='No current project selected.  ';
                   end;
          5 : begin
                 next_module:=os;
                 project_id:=0;
                 user_id:=0;
              end;
          end;
       until next_module<>undefined;
    end; { select_option }

{*****************************************************************}
{* Procedure : Execute_Next                                      *}
{* Parameters : Module, Project                                  *}
{* Global Variables : Current                                    *}
{* Entry Conditions : Module and Project will identify the       *}
{* module which is to be executed next.                          *}
{* Exit Conditions : The selected module will be executed        *}
{* after the Argument file is set up with the names of all       *}
{* of the files required by the module.  The Directory will      *}
{* have been updated to reflect any changes in the status of     *}
{* all the files used by the command processor.                  *}
{* Note that control will NOT return to the caller!              *}
{*****************************************************************}
procedure execute_next(next_mod : modules; project : integer);
    begin
       update_header(current);
       load_args(next_mod,project);
       updategd; { Update Global Directory and Close all files }
       exec(next_mod); { No direct return from this procedure! }
    end;

begin { command processor }
    init_all;
```

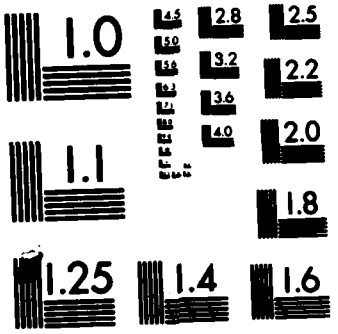
Command Processor Module

```
   read_args(current);
   with current do
   repeat
      if error_code<>0
         then resolve(error_code);
      if ((user_id=0) and (error_code=0))
         then get_identity(user_id,valid)
         else valid:=true;
      if (valid and (error_code=0))
         then select_option(user_id, project_id, completion, next_mod);
      if error_code=0
         then execute_next(next_mod, project_id);
   until error_code=0
end.
```

AD-A138 427 PRINTED CIRCUIT BOARD LAYOUT BY MICROCOMPUTER(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING E W KRAUSMAN DEC 83 AFIT/GE/EE/83D-35 3/3

UNCLASSIFIED F/G 9/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX E - LNW Source Code

```
program termio;
const q_x           = 5;
      q_y           = 12;
      q_lines       = 3;
      m_x           = 5;
      m_y           = 1;
      m_lines       = 10;
      h_x           = 30;
      h_y           = 1;
      h_lines       = 10;
      clear_to_eol = '#1E';
      max_prompt    = 30;

type boxes = (q, m, h);
     char_string = array[1..max_prompt] of char;

procedure gotoxy(x,y : integer); external;
procedure writech(ch : char); external;
function getkey : char; external;

procedure wait_key;
   var dummy : char;
   begin
      dummy:=getkey;
   end;

procedure goto_box(box : boxes; x,y : integer);
begin
   case box of
   q : gotoxy(q_x+x,q_y+y);
   m : gotoxy(m_x+x,m_y+y);
   h : gotoxy(h_x+x,h_y+y);
   end;
end;

procedure clear_line(box : boxes; y : integer);
begin
   goto_box(box,0,y);
   writech(clear_to_eol);
end;
```

TermIO Module

```
procedure clear_box(box : boxes);
var line, lines : integer;
begin
   case box of
   q : lines:=q_lines;
   m : lines:=m_lines;
   h : lines:=h_lines;
   end;
   for line:=0 to lines do clear_line(box,line);
end;

function get_count(box : boxes) : integer;
begin
   case box of
   q : get_count:=q_lines;
   m : get_count:=m_lines;
   h : get_count:=h_lines;
   end;
end;

procedure input_err(err_msg : char_string);
begin
   clear_line(m,m_lines+1);
   write(err_msg);
   write(' Press any key.');
   wait_key;
   clear_line(m,m_lines+1);
end;

begin
{$NULLBODY}
end.
```

QueryYN Module

```
program query_yes_no;
const
     cursoron = '#0E';
    cursoroff = '#0F';
       return = '#0D';
       escape = '#03'; {cntl C}
   max_prompt = 30;      { Maximum length of a prompt }

type
{ Query Type Definitions }
     prompt_string = packed array[1..max_prompt] of char;
             p_len = 1..max_prompt;
             yesno = (yes, no, i_dunno);
             boxes = (q, m, h);

procedure clear_box(box : boxes); external;
procedure goto_box(box : boxes; x,y : integer); external;
procedure writech(ch : char); external;
function getkey : char; external;
procedure help(help_index : integer); external;

procedure query_yn(prompt : prompt_string;
            prompt_length : p_len;
               var answer : yesno;
               help_index : integer);

   var    character : char;
                  i : p_len;
```

QueryYN Module

```
   begin { qyery_yn }
      answer:=i_dunno;
      clear_box(q); writech(cursoroff);
      goto_box(q,0,0);
      for i:=1 to prompt_length do
          writech(prompt[i]);
      goto_box(q,0,1);
      write('Please Respond with Yes or No.');
      repeat
         goto_box(q,prompt_length,0);
         character:=getkey;
         case character of
              'Y','y' : begin
                           answer:=yes;
                           write(' Yes          ')
                        end;
              'N','n' : begin
                           answer:=no;
                           write(' No           ')
                        end;
                  '?' : begin
                           answer:=i_dunno;
                           write(' I Don''t Know!');
                           help(help_index)
                        end;
      end;
      until (character=return);
      writech(cursoron);
      clear_box(q);
   end;

begin
{$NULLBODY}
end.
```

QueryNum Module

```
program query_numeric(input,output);
{ This is the numeric entry portion of the Query Module }
const
    leftarrow = '#08';  { Define arrow keys }
   rightarrow = '#09';
      uparrow = '#5B';
    downarrow = '#0A';
   cursorleft = '#18';  { Define cursor motion commands }
  cursorright = '#19';
   cursordown = '#1A';
     cursorup = '#1B';
     cursoron = '#0E';
    cursoroff = '#0F';
          ins = '#13'; {cntl S}   { Define Editing Commands }
          del = '#04'; {cntl D}
       change = '#15'; {cntl U}
       return = '#0D';
       escape = '#03'; {cntl C}
         null = '#00';
   max_prompt = 30;      { Maximum length of a prompt }
      max_num = 10;      { Maximum length of a number }

type
{ Query Type Definitions }
              unit = (mils, inches, mm, scalar);
     prompt_string = packed array[1..max_prompt] of char;
             p_len = 1..max_prompt;
             boxes = (q, m, h);

{ Screen I/O Declarations }
procedure goto_box(box : boxes; x,y : integer); external;
procedure clear_line(box : boxes; y : integer); external;
procedure clear_box(box : boxes); external;
procedure writech(ch : char); external;
procedure wait_key; external;
function getkey : char; external;
procedure help(help_index : integer); external;

procedure querynum(prompt : prompt_string;
            prompt_length : p_len;
                 min, max : integer;
              var answer : integer;
                   units : unit;
              help_index : integer);
```

QueryNum Module

```
   var range_violation,
          help_request,
           unit_change : boolean;
                     i : 1..max_prompt;
response, d_min, d_max : real;
                 dummy : char;

procedure getnumber;
   type pos = 1..max_num;
   var         character : char;
        point, negative : boolean;
          number_string : packed array[1..max_num] of char;
       current_position,
       point_position,i : pos;

   procedure show_number;
      begin
         goto_box(q,current_position+prompt_length+1,0);
         for i:=current_position to max_num do
             writech(number_string[i]);
         goto_box(q,current_position+prompt_length+1,0);
      end;

   procedure sign(ch : char);
      begin
        goto_box(q,prompt_length+1,0);
        writech(ch);
        goto_box(q,current_position+prompt_length+1,0)
     end;

   procedure add_digit;
      begin
         number_string[current_position]:=character;
         writech(character);
         if current_position<max_num
            then current_position:=current_position+1
            else writech(cursorleft)
      end;
```

QueryNum Module

```
procedure insert_digit;
   begin
      if point
         then if point_position>=current_position
                 then begin
                         point_position:=point_position+1;
                         if point_position>max_num
                            then point:=false
                      end;
      if current_position<max_num
         then for i:=max_num downto current_position+1 do
                 number_string[i]:=number_string[i-1];
      number_string[current_position]:=' ';
      show_number
   end;

procedure delete_digit;
   begin
      if point
         then begin
                 if current_position=point_position
                    then point:=false;
                 if point_position>current_position
                    then point_position:=point_position-1;
              end;
      if current_position<max_num
         then for i:=current_position to max_num-1 do
                 number_string[i]:=number_string[i+1];
      number_string[max_num]:=' ';
      show_number
   end;

procedure convert;
var power : real;
```

QueryNum Module

```
procedure get_int_part(position : pos);
   var i : integer;
   begin
      power:=1;
      for i:=position downto 1 do
          if(number_string[i]>='0' and
             number_string[i]<='9')
             then begin
                    response:=response+power*
                        (ord(number_string[i])-ord('0'));
                    power:=power*10
                  end;
   end;

procedure get_frac_part(position : pos);
   var i : integer;
   begin
      power:=0.1;
      for i:=position to max_num do
          if (number_string[i]>='0' and
              number_string[i]<='9')
              then begin
                     response:=response+power*
                        (ord(number_string[i])-ord('0'));
                     power:=power/10
                   end;
   end;

begin { convert }
   response:=0.0;
   if not point
      then get_int_part(max_num)
      else begin
             if point_position>1
                then get_int_part(point_position-1);
             if point_position<max_num
                then get_frac_part(point_position+1)
           end;
   if negative
      then response:=response*(-1);
end;
```

QueryNum Module

```
begin { getnumber }
   current_position:=1;
   for i:=1 to max_num do number_string[i]:=' ';
   show_number;
   help_request:=false; unit_change:=false; point:=false;
   negative:=false;
   repeat
      character:=getkey;
      case character of
        '0','1','2','3','4',
        '5','6','7','8','9',',' :
              begin
                 if (point and (current_position=point_position))
                    then point:=false;
                 add_digit
              end;
        '.' : if not point
                 then begin
                        point_position:=current_position;
                        add_digit; point:=true
                      end;
        '-' : begin
                 negative:=true;
                 sign('-')
              end;
        '+' : begin
                 negative:=false;
                 sign(' ')
              end;
        '?' : help_request:=true;
        ins : insert_digit;
        del : delete_digit;
     change : unit_change:=true;
  leftarrow : if current_position>1
                 then begin
                        current_position:=current_position-1;
                        writech(cursorleft)
                      end;
 rightarrow : if current_position<max_num
                 then begin
                        current_position:=current_position+1;
                        writech(cursorright)
                      end;
     return : convert;
      end { case }
   until (character=return or character='?' or character=change);
end; { getnumber }
```

QueryNum Module

```
function to_inches(mils : integer): real;
   begin to_inches:=mils/1000.0 end;

function to_mm(mils : integer): real;
   begin to_mm:=to_inches(mils)*25.4 end;

begin { querynum }
   clear_box(q);
   repeat
      range_violation:=false;
      clear_line(q,0);
      for i:=1 to prompt_length do writech(prompt[i]);
      for i:=1 to max_num do writech(' ');
      case units of
         inches: begin
                    write('   inches     ');
                    d_min:=to_inches(min);
                    d_max:=to_inches(max);
                    clear_line(q,1);
                    write('Range: ',d_min:7:3,' to ',d_max:7:3)
                 end;
           mm: begin
                    write('   millimeters');
                    d_min:=to_mm(min);
                    d_max:=to_mm(max);
                    clear_line(q,1);
                    write('Range: ',d_min:7:3,' to ',d_max:7:3)
                 end;
         mils: begin
                    write('   mils       ');
                    d_min:=min; d_max:=max;
                    clear_line(q,1);
                    write('Range: ',d_min:6:0,' to ',d_max:6:0)
                 end;
         scalar: begin
                    d_min:=min; d_max:=max;
                    clear_line(q,1);
                    write('Range: ',d_min:6:0,' to ',d_max:6:0)
                 end;
      end;
```

QueryNum Module

```
      getnumber;
      if unit_change
         then case units of
            scalar: units:=scalar;
              mils: units:=inches;
            inches: units:=mm;
                mm: units:=mils;
              end;
      if help_request
         then help(help_index);
      if (not (unit_change or help_request)
          and (response<d_min or response>d_max))
         then begin
                 clear_line(q,2);
                 write('Response not within range. Press any key to cont
inue');
                 wait_key;
                 clear_line(q,2);
                 range_violation:=true
              end;
   until (not (help_request or range_violation or unit_change));
   case units of
      mils,scalar : answer:=round(response);
            inches : answer:=round(response*1000);
                mm : answer:=round(response*1000/2.54);
   end;
   clear_box(q);
end;

begin
{$NULLBODY}
end.
```

QueryStr Module

```
program query_string;
{ This is the string entry portion of the query module. }
const
    leftarrow = '#08';  { Define arrow keys }
   rightarrow = '#09';
      uparrow = '#5B';
    downarrow = '#0A';
   cursorleft = '#18';  { Define cursor motion commands }
  cursorright = '#19';
   cursordown = '#1A';
     cursorup = '#1B';
     cursoron = '#0E';
    cursoroff = '#0F';
          ins = '#13'; {cntl S}   { Define Editing Commands }
          del = '#04'; {cntl D}
       return = '#0D';
       escape = '#03'; {cntl C}
         null = '#00';
   max_prompt = 30;      { Maximum length of a prompt }
      max_str = 30;      { Maximum length of a string }

type
{ Query Type Definitions }
     prompt_string = packed array[1..max_prompt] of char;
             p_len = 1..max_prompt;
          position = 1..max_str;
       string_case = (upper, lower, none);
       char_string = packed array[1..max_str] of char;
             boxes = (q, m, h);
{ Screen I/O Declarations }
procedure goto_box(box : boxes; x,y : integer); external;
procedure clear_box(box : boxes); external;
procedure writech(ch : char); external;
function getkey : char; external;

procedure help(help_index : integer); external;

procedure querystr(prompt : prompt_string;
           prompt_length : p_len;
                min, max : position;
              var answer : char_string;
       var string_length : position;
                make_case : string_case;
              help_index : integer);

   var help_request : boolean;
                   j : p_len;
                   i : position;

procedure getstring;
   var character : char;
    i, current_position : position;
```

QueryStr Module

```
procedure redisplay;
   var i : position;
   begin
      goto_box(q,prompt_length+current_position,0);
      for i:=current_position to string_length do
          writech(answer[i]);
      if string_length<max
         then for i:=string_length+1 to max do
              writech('_');
      goto_box(q,prompt_length+current_position,0);
   end;

procedure insert_char;
   var i : position;
   begin
      if ((current_position < max) and
          (string_length>=current_position))
         then begin
                for i:=max downto current_position+1 do
                    answer[i]:=answer[i-1];
                if string_length<max
                   then string_length:=string_length+1;
                answer[current_position]:=' ';
                redisplay
          end;
   end;

procedure delete_char;
   var i : position;
   begin
      if ((current_position <max) and
          (string_length>=current_position))
         then begin
                for i:=current_position to max-1 do
                    answer[i]:=answer[i+1];
                string_length:=string_length-1;
                answer[max]:=' ';
                redisplay
              end;
      if current_position=max and string_length=max
         then begin
                string_length:=string_length-1;
                answer[max]:=' ';
                writech('_'); writech(cursorleft)
              end;
   end;

procedure move_left;
   begin
      if current_position>1
         then begin
```

```
                    current_position:=current_position-1;
                    writech(cursorleft);
                 end;
      end;

   procedure move_right;
      begin
         if current_position<max
            then begin
                    current_position:=current_position+1;
                    writech(cursorright);
                 end;
      end;

   procedure add_char;
      begin
         if string_length<current_position
            then string_length:=current_position;
         answer[current_position]:=character;
         writech(character);
         if current_position<max
            then current_position:=current_position+1
            else writech(cursorleft);
      end;

    begin { getstring }
      goto_box(q,prompt_length+1,0);
      if max>max_str then max:=max_str;
      current_position:=1;
      string_length:=0;
      for i:=1 to max_str do answer[i]:=' ';
      help_request:=false;
      repeat
         character:=getkey;
         case character of
                ins : insert_char;
                del : delete_char;
         leftarrow : move_left;
         rightarrow : move_right;
             return : if string_length<min
                         then character:=null;
                '?' : help_request:=true;
         end;
         if character>=' ' and character<=chr(127)
            then add_char;
      until ((character=return) or help_request);
   end;

procedure make_upper;
   var i : position;
   begin
      for i:=1 to max do
```

```
            if answer[i] >= 'a' and answer[i] <= 'z'
               then answer[i]:=chr(ord(answer[i])-ord('a')+ord('A'));
   end;

procedure make_lower;
   var i : position;
   begin
      for i:=1 to max do
          if answer[i] >= 'A' and answer[i] <= 'Z'
             then answer[i]:=chr(ord(answer[i])-ord('A')+ord('a'));
   end;

begin { querystr }
   clear_box(q);
   repeat
      goto_box(q,0,0);
      for j:=1 to prompt_length do writech(prompt[j]);
      writech(' ');
      for i:=1 to max do writech('_');
      goto_box(q,0,1);
      write('Response must be between ',min:2, ' and ',max:2,' character
s');
      getstring;
      if help_request then help(help_index);
   until not help_request;
   case make_case of
        upper : make_upper;
        lower : make_lower;
   end;
   clear_box(q);
end;

begin
{$NULLBODY}
end.
```

DOSFile Module

```
{$widelist}
{*****************************************************************}
{* Operating System Dependent File Operations                    *}
{*****************************************************************}
program dos_file;
const doscmndi = #4405; {Address of DOSPLUS Command interpreter}
      dosrun   = #4433; {Address of DOSPLUS Run File Routine}
      return   = '#0D';

type byte     = 0..255;
     filespec = array[1..12] of char;
     drive_id = 0..1;

{The procedure call$ is used to call assembly language routines}
procedure call$(address : integer; var a,status : byte;
          var bc,de,hl,ix,iy : integer); external;

{See the Disk ID module for details on the following }
procedure switchdisk(drive : drive_id; disk_id : integer);
          external;

{*****************************************************************}
{* Procedure : Kill_File                                         *}
{* Parameters : Drive, Disk_ID, File_Name                        *}
{* Entry Conditions : The parameters identify the file to be     *}
{* wiped out.                                                    *}
{*****************************************************************}
procedure kill_file(drive : drive_id; disk_id : integer;
                    file_name : filespec);
   var command  : array[1..18] of char;  { Text of Command }
       cmndaddr : integer;               { Address of Command }
       ch       : 1..12;                 { Character pointer }
       db       : byte;                  { Dummy Byte }
       dw       : integer;               { Dummy Word }
   begin
      switchdisk(drive,disk_id);
      command:='KILL               ';
     { insert File_Name after the word 'KILL' in the command }
      for ch:=1 to 12 do command[ch+5]:=file_name[ch];
      command[18]:=return; cmndaddr:=location(command);
      call$(doscmndi,db,db,dw,dw,cmndaddr,dw,dw);
   end;

{*****************************************************************}
{* Procedure : Run_File - Execute a program                      *}
{* Parameters : File_Name, Drive, Disk_ID                        *}
{* Entry Conditions : The parameters identify an executable      *}
{* program that is to be run.  Control will not return!!!!!     *}
{*****************************************************************}
procedure run_file(file_name : filespec; drive : drive_id;
                   disk_id : integer);
   var dcb_addr : integer;              { Address of File DCB }
```

DOSFile Module

```
        dcb       : array[1..32] of char; { File Control Block }
        ch        : 1..12;                { Character Counter }
        db        : byte;                 { Dummy Byte }
        dw        : integer;              { Dummy Word }
    begin
       switchdisk(drive, disk_id);
       for ch:=1 to 12 do dcb[ch]:=file_name[ch];
       dcb[13]:=return;
       dcb_addr:=location(dcb);
       call$(dosrun,db,db,dw,dcb_addr,dw,dw,dw);
    end;

begin
{$nullbody}
end.
```

DiskID Module

```
{$WIDELIST}
program diskid;
const max_prompt = 30;      { Maximum length of a prompt }
         max_str = 30;      { Maximum length of a string }

type
{ Query Type Definitions }
     prompt_string = packed array[1..max_prompt] of char;
             p_len = 1..max_prompt;
             yesno = (yes, no, i_dunno);
             boxes = (q, m, h);

{ File Access Type Definitions }
          drive_id = 0..1; { or (A, B) }
            id_num = file of integer;
              byte = 0..255;

{ Screen I/O Declarations }
procedure clear_line(box : boxes; y : integer); external;
procedure wait_key; external;

{ Query Declarations }
procedure query_yn(prompt : prompt_string;
             prompt_length : p_len;
                var answer : yesno;
                help_index : integer); external;

procedure setacnm(var id_file:id_num; spec:string); external;
procedure io$error(newstate : boolean;
                   var oldstate : boolean); external;
function file$status(var f:id_num) : byte; external;

procedure set_id(drive : drive_id; var id_file : id_num);
   begin
      case drive of
          0 : setacnm(id_file,bldstr('disk/id:0'));
          1 : setacnm(id_file,bldstr('disk/id:1'));
    otherwise setacnm(id_file,bldstr('disk/id:1'));
      end;
   end;

function whichdisk(drive : drive_id) : integer;
   var id_file : id_num;
            id : integer;
           old : boolean;
             d : yesno;
        status : byte;

   begin  { whichdisk }
      io$error(false,old);
      set_id(drive,id_file);
      repeat
```

```
      reset(id_file); status:=file$status(id_file);
      if status=0
         then begin
                 read(id_file,id);
                 status:=file$status(id_file);
              end;
      if status=0 then whichdisk:=id;
      if status=8
         then repeat
              query_yn('Have you put a disk in yet?    ',27,d,21);
              until d=yes;
      if status=24 then whichdisk:=0;
      until status<>8;
      close(id_file);
   end;

procedure id_disk(drive:drive_id; disk_num:integer);
   var id_file : id_num;
   begin
      set_id(drive,id_file);
      rewrite(id_file);
      write(id_file,disk_num);
      close(id_file);
   end;

procedure switchdisk(drive : drive_id; disk_id : integer);
   var   answer : yesno;
             id : integer;

   procedure close_all(drive : drive_id); external;

   begin { switchdisk }
      id:=whichdisk(drive);
      if id<>disk_id
        then begin
             close_all(drive);
             repeat
              clear_line(q,-1);
              write('Insert disk number',disk_id:3,' into drive ',drive:
1);
              repeat
               query_yn('Ready to Go?                    ',12,
                        answer,100);
              until answer=yes;
              clear_line(q,-1);
              id:=whichdisk(drive);
              if id<>disk_id
                then begin
                       write('This is disk number',id:3,', not ',disk_id:
3);
                       write('! Press any key.'); wait_key;
                     end;
```

```
            until id=disk_id;
          end;
   end;

procedure new_disk(disk_id : integer);
   var answer : yesno;
   begin
      clear_line(q,-1);
      write('Put a blank formatted disk into drive 1');
      repeat
      query_yn('Are you ready?                ',14,answer,19);
      until answer=yes and whichdisk(1)=0;
      id_disk(1,disk_id);
      clear_line(q,-1);
      write('Label this disk as Disk ## ',disk_id:3);
      repeat
      query_yn('Have you labelled it yet?     ',25,answer,22);
      until answer=yes;
      clear_line(q,-1);
   end;

begin
{$NULLBODY}
end.
```

LinkFile Module

```
{$WIDELIST}
{*****************************************************************}
{* File Access Module - Controls access to all files           *}
{* ************************************************************}
{* All system files are maintained as two doubly-linked        *}
{* circular lists.  One list contains all of the allocated     *}
{* records in the file, and the other contains all of the      *}
{* free records.  Every record in the file must belong to one  *}
{* of these lists.  FNon linked files are also supported.      *}
{* For maximum portability, the maximum record length is 128   *}
{* bytes.                                                      *}
{*****************************************************************}
program file_access;
const
     end_of_list = -1;      { Indicates End of Linked List }
        max_open = 10;      { Maximum number of open files }
    max_rec_size = 128;     { Maximum Record Size }
       max_buffs = 2:       { Number of simultaneously open files }

type
{ File Access Type Definitions }
          drive_id = 0..1; { or (A, B) }
          filespec = packed array[1..12] of char;
         whichfile = 1..max_open;
             links = record
                       next : integer; { Record Number of next }
                       prev : integer; { Record Number of Last }
                     end;
           max_rec = record
                     case boolean of
                     true : (data : array[1..max_rec_size] of char);
                     false: (link : links);
                     end;
         data_file = file of max_rec;
         file_desc = record
                           fs : string; { File Name }
                       linked : boolean;
                        drive : drive_id;
                      on_line : boolean;
                      rec_len, { Record Length }
                      disk_id, { Diskette containing file }
                       status, { System Dependent File Status }
                        first, { Record Number of First Entry }
                    next_free, { Record Number of Free List }
                   recs_avail, { Number of Unused Records }
                    next_read, { Next to be Read }
                    last_read, { Last Record Accessed }
                      buf_num : 0..max_buffs;{ Current Buffer NUmber }
                     end;
             where = record
                       linked    : boolean;
                       file_name : filespec;
```

```
                        drive : drive_id;
                        disk_id,
                        rec_len,
                        recs_avail,
                        first,
                        free : integer;
                      end;

{ Global Arrays containing all required File Information }
common files : array[whichfile] of file_desc;
       P_buff : array[1..max_buffs] of data_file;
       P_slot : array[1..max_buffs] of byte;
       next_1 : byte;
       buffer : max_rec;

{ System Dependent Random Access File Routines }
procedure openrand(var f:data_file; recordlen:integer;
                   pathname:string; var status:integer); external;
procedure readrand(var f:data_file; recordnum:integer;
          var dat:max_rec; var status:integer); external;
procedure writerand(var f:data_file; recordnum:integer;
          var dat:max_rec; var status:integer); external;
procedure closerand(var f:data_file); external;

procedure fread(file_num : whichfile; rec_num : integer;
                var buffer : max_rec; linkonly : boolean);

   var temp_buff : max_rec;
   begin
      with files[file_num] do
      begin
         if linkonly
            then begin
                 readrand(P_buff[buf_num],rec_num,temp_buff,status);
                 buffer.link:=temp_buff.link;
                 end
            else readrand(P_buff[buf_num],rec_num,temp_buff,status);
      end;
   end;

procedure fwrite(file_num : whichfile; rec_num : integer;
                 var buffer : max_rec; linkonly : boolean);

   var temp_buff : max_rec;
   begin
      with files[file_num] do
      begin
         if linkonly
            then begin
                 readrand(P_buff[buf_num],rec_num,temp_buff,status);
                 temp_buff.link:=buffer.link;
                 writerand(P_buff[buf_num],rec_num,temp_buff,status);
```

```
                end
            else writerend(P_buff[buf_num],rec_num,buffer,status);
      end;
   end;

{ See the Disk ID Module for details on the following }
procedure switchdisk(drive : drive_id;
                     diskid : integer); external;

{*****************************************************************}
{* Procedure : StateF - Determine Current Pointer Values       *}
{* Parameters : File_Num, First, Free                          *}
{* Entry Conditions : File_Num identifies the file to be       *}
{* looked at.                                                  *}
{* Exit Conditions : The current values of the heads of the    *}
{* allocated and free lists are assigned to First and Free.    *}
{*****************************************************************}
procedure statef(file_num:whichfile; var first,free : integer);
   access files; { Global Array }
   begin
     first:=files[file_num].first;
     free:=files[file_num].next_free;
   end;

{*****************************************************************}
{* Function : RoomF - How much Room is Left in the File?       *}
{* Parameters : File_Num                                       *}
{* Result : Number of Unallocated Records in the file          *}
{*****************************************************************}
function roomf(file_num : whichfile) : integer;
   access files;
   begin
      roomf:=files[file_num].recs_avail;
   end;

{*****************************************************************}
{* Procedure : ResetF - Set pointers to the top of the file    *}
{* Parameters : File_Num                                       *}
{* Process : The file pointers Next_Read and Last_Read are     *}
{* set to the first allocated record in the file.  Any file    *}
{* access after a resetf will access the first record.         *}
{*****************************************************************}
procedure resetf(file_num : whichfile);
   access files;
   begin
      with files[file_num] do
      if linked
         then begin
                 last_read:=first;
                 next_read:=first;
              end
         else next_read:=0;
```

```
   end;

{*****************************************************************}
{* Procedure : InitF - Initialize a slot in the Files array    *}
{* Parameters : File_Num, File_Loc                             *}
{* Entry Conditions : File_Num identifies the slot to be       *}
{* initialized, and File_Loc contains the initialization       *}
{* parameters.                                                 *}
{* Process : The File_Num slot will be loaded with File_Loc    *}
{* and the file will be set to off_line status.  The file      *}
{* will then be reset to the first record.                     *}
{*****************************************************************}
procedure initf(file_num : whichfile; file_loc : where);
   access files;
   begin
      with files[file_num] do
      begin
       { System Dependent String Manipulation Below! }
       { TRS-80 Strings are manitained on the heap.  To recover
         the memory they use, they must be Disposed before new
         values are assigned to them.  Bldstr just converts a
         literal or character array into a string on the heap. }
       if fs<>nil then dispose(fs);
       fs:=bldstr(file_loc.file_name);
       { Copy the rest of File_Loc to slot File_Num }
       linked:=file_loc.linked;
       disk_id:=file_loc.disk_id;
       drive:=file_loc.drive;
       rec_len:=file_loc.rec_len;
       recs_avail:=file_loc.recs_avail;
       first:=file_loc.first;
       next_free:=file_loc.free;
       { Set file off-line }
       on_line:=false;
      end;
      resetf(file_num);
   end;

{*****************************************************************}
{* Procedure : Init_Files -Onetime Files Array Initialization  *}
{* Process : Sets ALL file slots off-line so that a Close_All  *}
{* operation (See Below) will not attempt to close file slots  *}
{* that were never initialized. Should only be called once.    *}
{*****************************************************************}
procedure init_files;
   access files,x;
   var i : 1..max_open;
   begin
      for i:=1 to max_open do
          with files[i] do
          begin
             on_line:=false;
```

```
            fs:=nil;
         end;
      for i:=1 to max_buffs do P_slot[i]:=0;
      next_1:=1;
   end;

{*****************************************************************}
{* Procedure : CloseF - Close Random Access File                 *}
{* Paramaters : File_Num                                         *}
{* Process : The file in slot File_Num is closed and the On-     *}
{* Line flag is reset. If already off-line, nothing is done.     *}
{*****************************************************************}
procedure closef(file_num : whichfile);
   access files;
   begin
      with files[file_num] do
      begin
      if on_line
         then begin
                closerand(P_buff[buf_num]);
                on_line:=false;
                P_slot[buf_num]:=0; { Release the buffer }
              end;
      end;
   end;

{*****************************************************************}
{* Procedure : OpenF - Open file for reading or writing          *}
{* Parameters : File_Num                                         *}
{* Process : The file in slot File_Num is brought on-line.       *}
{* If the file is already On-Line, then nothing need be done.    *}
{* If, however, the file is off-line, then the diskette          *}
{* containing the file must be mounted, the system dependent     *}
{* association between logical and physical files must be made   *}
{* and the file opened for random access.                        *}
{*****************************************************************}
procedure openf(file_num : whichfile);
  access files;
  var i : integer;
  begin
     with files[file_num] do
        if not on_line then
           begin
              { Make sure the right disk is present }
              switchdisk(drive,disk_id);
              { Look for an available buffer }
              buf_num:=0; i:=1;
              repeat
                 if P_slot[i]=0 then buf_num:=i;
                 i:=i+1;
              until((i>max_buffs) or (buf_num<>0));
              { If one isn't available, free one up. }
```

```
                if buf_num=0
                   then begin
                           closef(P_slot[next_1]);
                           buf_num:=next_1;
                           next_1:=next_1+1;
                           if next_1>max_buffs then next_1:=1;
                        end;
                { Reserve the buffer for this slot }
                P_slot[buf_num]:=file_num;
                { Open the file }
{ Alcor Pascal requires non-simple files to be initialized
  before they can be accessed.  The :: is a type transfer
  operator that tells the compiler to treat the variable as if
  it was of the specified type, integer in this case. }
                P_buff[buf_num]::integer:=0;
               { System Dependent Random Access File Open }
                openrand(P_buff[buf_num],rec_len,fs,status);
                on_line:=true;
             end;
  end;

{*****************************************************************}
{* Procedure : Close_All - Close all files on a drive            *}
{* Parameters : Drive                                            *}
{* Entry Conditions : Drive indicates which drive is to have     *}
{* all of its open files closed.                                 *}
{* Process : This routine is called prior to removal of a disk   *}
{* from a drive to  ensure file integrity.  Each slot in Files   *}
{* is checked to see if the drive matches.  If it does, the      *}
{* file is closed.                                               *}
{*****************************************************************}
procedure close_all(drive : drive_id);
   access files;
   var fx : whichfile;
   begin
      for fx:=1 to max_open do
        if files[fx].drive=drive then closef(fx)
   end;

{*****************************************************************}
{* Procedure : ReadF - Read the Next Record                      *}
{* Parameters : File_Num, Buffer, EOLF                           *}
{* Entry Conditions : File_Num identifies to file to read.       *}
{* Next_Read contains the record number of the next record to    *}
{* be accessed.                                                  *}
{* Exit Conditions : Buffer contains the record read, and        *}
{* Next_Read and Last_Read will be updated accordingly, unless   *}
{* an end of file condition was detected.  In this case, the     *}
{* Buffer will not be modified and EOLF will be set.             *}
{*****************************************************************}
procedure readf(file_num : whichfile; var buffer : links;
                var eolf : boolean);
```

LinkFile Module

```
   access files;
   var link : links;
   begin
      openf(file_num);
      with files[file_num] do
      if linked
        then begin
                { Check for end of list condition }
                if ((next_read=first) and (last_read<>first)) or
                    (next_read=end_of_list)
                then eolf:=true
                else begin
                        fread(file_num,next_read,buffer,false);
                        last_read:=next_read;
                        next_read:=buffer.link.next;
                        if last_read=next_read
                           { There is only one record }
                        then eolf:=true
                        else eolf:=false;
                     end;
             end
        else begin
                fread(file_num,next_read,buffer,false);
                next_read:=next_read+1;
                if status=0
                   then eolf:=false
                   else eolf:=true;
             end;
   end; { readf }

{*****************************************************************}
{* Procedure : WriteF - Update the Last_Record accessed        *}
{* Parameters : File_Num, Buffer, WriteOK                      *}
{* Entry Conditions : File_Num identifies the file to write    *}
{* to.  Buffer contains the information to be written.  Last_ *}
{* Read is the record number to be written to.                 *}
{* Exit Conditions : Buff will have been written to the file. *}
{* No re-positioning will occur, so successive writes will     *}
{* over-write the same record.  WriteOK will be TRUE if there *}
{* were no write errors.                                       *}
{*****************************************************************}
procedure writef(file_num : whichfile; var buffer : links;
                 var writeok : boolean);
   access files;
   begin
      openf(file_num);
      with files[file_num] do
        if linked
           then if last_read<>end_of_list
                   then begin
                           writeok:=true;
                           fread(file_num,last_read,buffer,true);
```

```
                        fwrite(file_num,last_read,buffer,false);
                     end;
                  else writeok:=false
            else begin
                  fwrite(file_num,next_read,buffer,false);
                  next_read:=next_read+1;
                  if status=0
                     then writeok:=true
                     else writeok:=false;
               end;
   end; { writef }

{*****************************************************************}
{* Function : Del_Rec - Delete a record from one of the lists *}
{* Parameters : File_Num, Pointer                                *}
{* Result : Record number of deleted record                      *}
{* Entry Conditions : File_Num indicates file to use, Pointer *}
{* is the record number to be deleted.                           *}
{* Exit Conditions : The record will be deleted from the list *}
{* and pointer will be set to the the next record in the list.*}
{*****************************************************************}
function del_rec(file_num : whichfile;
           var pointer : integer) : integer;
   access files;
   var link : links;
   begin
      openf(file_num);
      with files[file_num] do
      begin
         del_rec:=pointer;
         { Read the Next and Previous Record numbers }
         fread(file_num,pointer,buffer,true);
         link:=buffer.link;
         if link.next=pointer { It's the last record }
            then pointer:=end_of_list
            else begin
                 { Make the Next record the new Current record }
                   pointer:=link.next;
                 { Delete the record from the forward list }
                   fread(file_num,link.prev,buffer,true);
                   buffer.link.next:=link.next;
                   fwrite(file_num,link.prev,buffer,true);
                 { Delete the record from the backward list }
                   fread(file_num,link.next,buffer,true);
                   buffer.link.prev:=link.prev;
                   fwrite(file_num,link.next,buffer,true);
                 end;
      end; { with }
   end; { del rec }

{*****************************************************************}
{* Procedure : Ins_Rec - Insert a record into one of the lists*}
```

LinkFile Module

```
{* Parameters : File_Num, New_Rec, Pointer                          *}
{* Entry Conditions : File_Num indicates which file to use,        *}
{* New_Rec is the number of the record to be inserted into the*}
{* list, and Pointer is the number of the record that New_Rec *}
{* will be inserted after.                                         *}
{* Exit Conditions : The links will have been adjusted so that*}
{* New_Rec logically follows Pointer in the list.                  *}
{*****************************************************************}
procedure ins_rec(file_num : whichfile; new_rec : integer;
                  pointer : integer);
   access files;
   var link : links;
   begin
      openf(file_num);
      with files[file_num] do
      begin
       if pointer=end_of_list  { Empty List }
          then begin
               { Make New_Rec the only entry in the list }
                buffer.link.prev:=new_rec;
                buffer.link.next:=new_rec;
                fwrite(file_num,new_rec,buffer,true);
               end
       else begin
            { Insert New_Rec into the forward list }
             fread(file_num,pointer,buffer,true);
             link.prev:=pointer; link.next:=buffer.link.next;
             buffer.link,next:=new_rec;
             fwrite(file_num,pointer,buffer,true);
            { Insert New_Rec into the backward list }
             fread(file_num,link.next,buffer,true);
             buffer.link.prev:=new_rec;
             fwrite(file_num,link.next,buffer,true);
            { Make New_Rec point to its neighbors }
             buffer.link:=link;
             fwrite(file_num,new_rec,buffer,true);
            end;
      end; { with }
   end; { ins_rec }

{*****************************************************************}
{* Procedure : InsertF - Insert a record into the file           *}
{* Parameters : File_Num, Rec_Num, Buffer                        *}
{* Entry Conditions : File_Num indicates the file to use.        *}
{* Buffer contains the information to be inserted.  Last_        *}
{* Record is the number of the record which Buffer is to be      *}
{* inserted after.                                               *}
{* Exit Conditions : Rec_Num will be the record number which     *}
{* was assigned to Buffer in the file.  Once a record is added*}
{* to the file, its position will never change.                  *}
{*****************************************************************}
procedure insertf(file_num : whichfile; var rec_num : integer;
```

```
                    var buffer : links);
   access files;
   begin
      openf(file_num);
      with files[file_num] do
      begin
         if next_free<>end_of_list
         then begin { There IS a free record }
                 recs_avail:=recs_avail-1;
                { Delete a record from the free list }
                 rec_num:=del_rec(file_num,next_free);
                { And add it to the allocated list }
                 ins_rec(file_num,rec_num,last_read);
                { Update the record pointers }
                 last_read:=rec_num;
                 if first=end_of_list
                 then begin {  This is the first record }
                        next_read:=last_read;
                        first:=last_read;
                      end;
                { Write Buffer to the file }
                 fread(file_num,last_read,buffer,true);
                 fwrite(file_num,last_read,buffer,false);
              end;
      end; { with }
   end; { insertf }

{*****************************************************************}
{* Procedure : DeleteF - Delete a record from a file           *}
{* Parameters : File_Num                                       *}
{* Entry Conditions : File_Num is the file to be used.  Last_  *}
{* Read is the record to be deleted.                           *}
{* Exit Conditions : Last_Read will point to the record foll-  *}
{* owing the deleted one.                                      *}
{*****************************************************************}
procedure deletef(file_num : whichfile);
   access files;
   var rec_num : integer;
   begin
      with files[file_num] do
        begin
           if first<>end_of_list
              then
               begin { There is a record to delete }
                  recs_avail:=recs_avail+1;
                  { Delete the record form the file }
                  rec_num:=del_rec(file_num,last_read);
                  if last_read=end_of_list
                     then begin
                             first:=end_of_list;
                             next_read:=end_of_list;
                          end;
```

```
                    { Add the deleted record to the free list }
                    ins_rec(file_num,rec_num,next_free);
                    next_free:=rec_num;
                end;
        end;
    end;

{*****************************************************************}
{* Procedure : Posf - Position file pointers                     *}
{* Parameters : file_num, Rec_num                                *}
{* Entry Conditions : File_Num indicates the slot to use, and    *}
{* Rec_num is the record number to which the file pointers       *}
{* will be set.                                                  *}
{*****************************************************************}
procedure posf(file_num : whichfile; rec_num : integer);
begin
    with files[file_num] do
    begin
        if ((rec_num=first) and (not linked))
        then resetf(file_num)
        else next_read:=rec_num;
    end;
end;

{*****************************************************************}
{* Procedure : CreateF - Create a new Index for the file         *}
{* Paramaters : File_Num, How_Many                               *}
{* Entry Conditions : File_Num indicates which file to use,      *}
{* and How_Many indicates how many records to allocate to the    *}
{* file.                                                         *}
{* Process : The links will be initialized as follows : all of   *}
{* the records will be assigned to the Free list, and the        *}
{* allocated list will be empty.  None of the information in     *}
{* the file will be erased, but it will not be accessable.       *}
{*****************************************************************}
procedure createf(file_num : whichfile; how_many : integer);
    access files;
    var rec : integer;
    begin
        openf(file_num);
        with files[file_num] do
        begin
            first:=end_of_list; next_free:=0;
            recs_avail:=how_many;
            for rec:=0 to how_many-1 do
            begin
                { Set up the forward and backward links }
                buffer.link.next:=rec+1; buffer.link.prev:=rec-1;
                { Make the next record of the last entry point
                  to the First entry }
                if buffer.link.next=how_many then buffer.link.next:=0;
                { Make the previous record of the first entry
```

```
                point to the Last entry - a circular list. }
              if buffer.link.prev=-1 then buffer.link.prev:=how_many-1;
              fwrite(file_num,rec,buffer,true);
            end;
        end;
        closef(file_num);
    end;

procedure erasef(file_num : whichfile);
    access files;
    var file_name : filespec;

    procedure kill_file(drive : drive_id; disk_id : integer;
                file_name : filespec); external;

    begin { erasef }
      with files[file_num] do
       getstr(fs,file_name); kill_file(drive,disk_id,file_name);
    end;

begin
{$NULLBODY}
end.
```

Menu Module

```
{$WIDELIST}
{*****************************************************************}
{* Menu Module - procedures for the manipulation of menus      *}
{*****************************************************************}
{* Menus may either be read directly from a file, or created   *}
{* dynamically.  In either case, the format of the menu in     *}
{* memory is identical. The selection of an item by the user   *}
{* is done by moving the cursor next to the desired item and   *}
{* pressing <RETURN>.  Help is available for each item by      *}
{* pressing <?>.  The menu structure allows for more than one  *}
{* level.  If the item selected has sub-selections, control    *}
{* will not return to the user until a terminal item is selec-*}
{* ted.                                                        *}
{*****************************************************************}
program menu_module;
const
      uparrow = '#5B';
    downarrow = '#0A';
   cursorleft = '#18';  { Define cursor motion commands }
  cursorright = '#19';
   cursordown = '#1A';
     cursorup = '#1B';
     cursoron = '#0E';
    cursoroff = '#0F';
       return = '#0D';
       escape = '#03'; {cntl C}
         null = '#00';
    max_item = 20;      { Maximum length of a menu item }
       menus = 2;       { Menu file is file number two }
type
{ Menu Type Definitions }
      menu_string = packed array[1..max_item] of char;
         item_ptr = ^menu_item;
     { In Memory Menu structure }
        menu_item = record
                      item_text : menu_string; {Text User sees}
                      item_code,               {Code returned }
                      help_index : integer;
                      next_item,               { Pointer }
                      previous_item,           { Pointer }
                      next_level : item_ptr;   { Pointer }
                    end;
      { Menu File Record Structure }
          menu_ln = record
                      menu_number : integer;  { Menu ID }
                      bump_down,        { True if item has
                                          Sub-selections }
                      bump_up : boolean;{ True if last item of
                                          current level }
                      item_text : menu_string;
                      item_code,
                      help_index : integer;
```

```
                    end;
        whichfile = 1..10;
            boxes = (q, m, h);     { Query, Menu, or Help Box }

{ Define current menu environment }
common  old : integer;     { Last menu accessed }
 first_item : item_ptr;    { Top of last menu }
   top_list : item_ptr;    { Top of Current List }
       list : item_ptr;    { Current position inside list }
      count : integer;     { Number of lines in current level }
  max_lines : integer;     { Number of lines allowed in a menu }

{ See TermIO Module for details on the following }
{ Screen I/O Declarations }
procedure goto_box(box : boxes; x,y : integer); external;
procedure clear_line(box : boxes; y : integer); external;
procedure clear_box(box : boxes); external;
function get_count(box : boxes) : integer; external;
procedure writech(ch : char); external;
function getkey : char; external;

{*****************************************************************}
{* Procedure : Erase_Menu - Delete a menu structure from Heap *}
{* Parameters : First_Item                                       *}
{* Entry Conditions : First_Item points to the first item of  *}
{* the menu structure currently defined.                         *}
{* Process : Each item of the current level will be Disposed. *}
{* If the menu menu item has a sub-level, however, Erase_Menu *}
{* will be recursively called to erase that level first.      *}
{* This will insure that the entire tree structure is erased. *}
{*****************************************************************}
procedure erase_menu(first_item : item_ptr);
   var current_item : item_ptr;
   begin
     { Make sure there is something to erase! }
      if first_item<>nil
      then repeat
             { Erase all lower levels of menu }
               if first_item^.next_level<>nil
                  then erase_menu(first_item^.next_level);
               current_item:=first_item;
             { Point to the next item }
               first_item:=first_item^.next_item;
               dispose(current_item);
           until first_item=nil;
   end;

{ See the LinkFile Module for details on the following }
procedure resetf(file_num : whichfile); external;
procedure readf(file_num : whichfile; var buffer : menu_ln;
                var eolf : boolean); external;
```

Menu Module

```
{ See the Help Module for details on the following }
procedure help(help_index:integer); external;
{*****************************************************************}
{* Function : Read_Menu - Read a Menu from the Menu File         *}
{* Parameters : Menu_Number                                      *}
{* Result : Pointer to first item of Menu                        *}
{* Entry Conditions : Menu_Number identifies which menu to       *}
{* read.                                                         *}
{* Exit Conditions : The menu will have been read from the       *}
{* Menu File into memory.  The result of the function is a       *}
{* pointer to the first item of the menu.                        *}
{*****************************************************************}
function read_menu(menu_number : integer) : item_ptr;
   var menu_line : menu_ln;     { A Line from the file }
            item : item_ptr;
            eolf : boolean;

{*****************************************************************}
{* Procedure : Read_Level - Read all menu items at current       *}
{*                          level into memory                    *}
{* Parameters : Last_Item, Last_Line                             *}
{* Global Variables : EOLF                                       *}
{* Entry Conditions : Last_Item points to an allocated but       *}
{* not yet initialized menu item.  Last_Line is the last item    *}
{* read from the file. EOLF is the end of file indicator.        *}
{* Process : First, the values in Last_Line are assigned to      *}
{* Last_Item.  Then, a test is made to see if there are sub-     *}
{* items under this one. If there are, then Read_Level is        *}
{* called recursively to read it.  Finally, a test is made to    *}
{* see if this item is the last item at this level.  If it is,   *}
{* then control will return to the caller.                       *}
{*****************************************************************}
   procedure read_level(last_item : item_ptr;
                        last_line : menu_ln);
      var level,item : item_ptr;
           menu_line : menu_ln;
                done : boolean;
      begin
         done:=false;
         repeat
           { Assign File values to Memory Menu }
           last_item^.item_text:=last_line.item_text;
           last_item^.item_code:=last_line.item_code;
           last_item^.help_index:=last_line.help_index;
           if last_line.bump_down
              then begin { There ARE sub-items }
                   { Allocate a new item and point to it }
                     new(level); last_item^.next_level:=level;
                   { Make it the first item of the next level}
                     level^.previous_item:=nil;
                   { Pre-read the Menu File }
                     readf(menus,menu_line,eolf);
```

```
                    { Read in the rest of this level }
                      read_level(level,menu_line);
                    end
                 else last_item^.next_level:=nil;
              if eolf or last_line.bump_up
                 then begin { Done with this level }
                        last_item^.next_item:=nil;
                        done:=true
                    end
                 else begin
                      { Read in the next item for this level }
                        readf(menus,menu_line,eolf);
                      { Allocate storage and point to it }
                        new(item); last_item^.next_item:=item;
                        item^.previous_item:=last_item;
                      { Set up for next iteration }
                        last_item:=item; last_line:=menu_line
                    end;
           until done;
        end;

    begin { read_menu }
       resetf(menus);
     { First, find the appropriate Menu in the Menu File }
       repeat
          readf(menus,menu_line,eolf);
       until menu_line.menu_number=menu_number;
     { Allocate storage for and point to the first item }
       new(item); item^.previous_level:=nil;
       read_menu:=item;
     { Read in the entire menu structure }
       read_level(item,menu_line);
   end;

{*****************************************************************}
{* Function : Select_Menu_Item                                   *}
{* Parameters : Menu_Index                                       *}
{* Result : Item_Code of the selected terminal item.             *}
{* Entry Conditions : Menu_Index points to the first item of     *}
{* a Menu currently in memory.                                   *}
{* Exit Conditions : The user will have selected one of the      *}
{* terminal menu items (an item with no sub-items).  Its code    *}
{* will be returned as the value of the function.  If a non-     *}
{* terminal item is selected, then a recursive call to this      *}
{* function will be made, until a terminal item is selected.     *}
{*****************************************************************}
function select_menu_item(menu_index : item_ptr) : integer;
   var index : item_ptr;
       character : char;
       selection : integer;

{*****************************************************************}
```

Menu Module

```
{* Procedure : Menu_Display - Display all Menu choices at the *}
{*                            current level                    *}
{* Process : The menu box area of the screen is cleared, and  *}
{* the choices available at the current level of the menu are *}
{* displayed on sequential lines of the screen.  The cursor is*}
{* positioned to the left of the first item.                  *}
{************************************************************}
   procedure menu_display;
      var index : item_ptr;
              l : integer;
      begin
         clear_box(m);
         l:=0;
         index:=menu_index;
         repeat
          { Indent each item to leave room for the cursor }
            goto_box(m,2,l);
            write(index^.item_text);
          { Point to the next item at this level }
            l:=l+1; index:=index^.next_item;
         until index=nil;
         goto_box(m,0,0);
         writech('*'); writech(cursorleft);
         writech(cursoroff);
      end;

   begin { Select Menu Item }
      menu_display;
      index:=menu_index;
      repeat
        character:=getkey;
        case character of
            uparrow : if index^.previous_item<>nil
                      then begin
                              writech(' '); writech(cursorleft);
                              writech(cursorup);
                              writech('*'); writech(cursorleft);
                              index:=index^.previous_item
                           end;
          downarrow : if index^.next_item<>nil
                      then begin
                              writech(' ');writech(cursorleft);
                              writech(cursordown);
                              writech('*'); writech(cursorleft);
                              index:=index^.next_item
                           end;
                '?' : begin
                         help(index^.help_index);
                         writech(cursoroff)
                      end;
             return : if index^.next_level=nil
                      then select_menu_item:=index^.item_code
```

```
                    else begin  { recursive call }
                selection:=select_menu_item(index^.next_level);
                        if selection=0
                           then begin
                            { Return from lower level with
                              no selection, so redisplay
                              current level }
                              character:=null;
                              menu_display;
                              index:=menu_index
                            end
                           else select_menu_item:=selection;
                        end;
            escape : select_menu_item:=0;
          end; { case }
       until character=return or character=escape;
       writech(cursoron);
       clear_box(m);
   end; { select_menu_item }

{*****************************************************************}
{* Function : Menu - Make a selection from a File Menu          *}
{* Parameters : Current                                         *}
{* Global Variables : Old, First_Item                           *}
{* Result : Item_Code of selected terminal item.                *}
{* Entry Conditions : Current is the number of the menu to be   *}
{* made the new current menu.  Old is the number of the last    *}
{* menu accessed, and First_Item points to the old menu.        *}
{* Exit Conditions : Old is updated to Current, and the         *}
{* function returns the selected code.                          *}
{*****************************************************************}
function menu(current : integer) : integer;
   access old, first_item;
   begin
      if old <> current
      then begin { A new menu must be read from the file }
              erase_menu(first_item);
              first_item:=read_menu(current);
              old:=current;
           end;
      menu:=select_menu_item(first_item);
   end;

{*****************************************************************}
{* Procedure : Init_List - Initialize a Memory List to empty    *}
{* Process : A list is to be built in memory.  It must first    *}
{* be initialized.  Any old list will be erased, and the size   *}
{* of the list displayed will be set to fill the menu-box.      *}
{*****************************************************************}
procedure init_list;
   access top_list, list, count, max_lines;
   begin
```

```
      erase_menu(top_list);
      top_list:=nil; list:=nil; count:=0;
    { Get the size of the Menu Box }
      max_lines:=get_count(m);
   end;

{*****************************************************************}
{* Procedure : Build_List - Add an item to the Memory List     *}
{* Parameters : Item_Text, Item_Code                           *}
{* Global Variables : Top_List, List, Count, Max_Lines         *}
{* Entry Conditions : Item_Text and corresponding Item_Code    *}
{* are to be added to the list in memory.                      *}
{* Process : The item will be added to the list.  If the       *}
{* number of items in the current level exceeds the capacity   *}
{* of the menu-box, then a new level will be created.  Count   *}
{* maiantains the number of items in the current level, and    *}
{* Max_Lines is the size of the menu-box.  Both are            *}
{* initialized by Init_List. The first call to this procedure *}
{* will define Top_List, and position for subsequent calls     *}
{* will be maintained by List.                                 *}
{*****************************************************************}
procedure build_list(item_text:menu_string; item_code:integer);
   access top_list, list, count, max_lines;
   var entry : item_ptr;
         c : integer;
   begin
      new(entry);
      if list<>nil
      then begin  { This isn't the first element of the list }
             list^.next_item:=entry;
             entry^.previous_item:=list;
           end
      else begin  { This IS the first element of the list }
             entry^.previous_item:=nil;
             top_list:=entry;
           end;
      if count=(max_lines-1)
      then begin  { Start a new level of menu }
             entry^.next_item:=nil;
             new(list);
             list^.previous_item:=nil;
             list^.next_level:=nil;
             entry^.next_level:=list;
             count:=0;
             entry^.item_text:='Rest of the list     ';
             entry^.help_index:=17;
           end
      else begin  { Add to present level of menu }
             entry^.next_level:=nil;
             list:=entry;
           end;
      list^.item_text:=item_text;
```

```
      list^.item_code:=item_code;
      list^.help_index:=18;
      list^.next_item:=nil;  { In case this is the last one }
      count:=count+1;
   end; { build list }

{*****************************************************************}
{* Function : Select_List - Select an item from Memory List    *}
{* Global Variables : Top_List                                  *}
{* Result : Item_Code of selected item.                         *}
{* Process : After a list hes been built in memory through      *}
{* calls to Build_List, an item may be selected with this       *}
{* function.  Since the Structure of both lists and menus is    *}
{* identical, Select_Menu_Item is used to do the actual work.   *}
{*****************************************************************}
function select_list : integer;
   access top_list;
   begin
      select_list:=select_menu_item(top_list);
   end;

{*****************************************************************}
{* Procedure : Init_Menu - Initialize Global Variables          *}
{* Global Variables : Old, First_Item, Top_List                 *}
{* Process : Prior to the first use of either a menu or a       *}
{* list, the pointers must be initialized.                      *}
{*****************************************************************}
procedure init_menu;
   access old,first_item,top_list;
   begin
      old:=0; first_item:=nil; top_list:=nil;
   end;

begin
{$NULLBODY}
end.
```

Help Module

```
{$WIDELIST}
{*****************************************************************}
{* Help Module - Provides information to the user about the    *}
{* system.  The help messages are stored in the Help File, and*}
{* are indexed.  Each query or menu item has been assigned a  *}
{* help index, which is used to look up the associated help   *}
{* information.                                                *}
{*****************************************************************}
program helper;
const max_help = 30;     { Maximum length of a help message line}
      max_lines = 10;    { Maximum number of help message lines }
      helps = 1;         { Help file is file number one }

type help_msg = record
                   help_index : integer;
                   line : packed array[1..max_help] of char;
                end;
     whichfile = 1..10;
         boxes = (q, m, h);

{ See the TermIO Module for details on the following }
{ Screen I/O Declarations }
procedure gotoxy(x,y : integer); external;
procedure clear_line(box : boxes; y : integer); external;

{ See the LinkFile Module for details on the following }
{ File Access Declarations }
procedure readf(file_num : whichfile; var buffer : help_msg;
                var eolf : boolean); external;
procedure resetf(file_num : whichfile); external;

{*****************************************************************}
{* Procedure : Help                                            *}
{* Parameters : Help_Index                                     *}
{* Entry Conditions : Help_Index identifies which help message*}
{* to look up.                                                 *}
{* Process : The Help File is scanned until a matching index  *}
{* is found.  This line and all subsequent lines are displayed*}
{* in the help-box area of the screen, until a line with a    *}
{* different index is found.                                   *}
{*****************************************************************}
procedure help(help_index : integer);
   var x,y : integer;
       mess : help_msg;
       eolf : boolean;
       l : 0..max_lines;
   begin
      resetf(helps);
    { Find the first line of the help message, if there is one.}
      repeat
         readf(helps,mess,eolf);
      until (mess.help_index=help_index) or eolf;
```

```
      l:=0;
    { Display all lines with matching index numbers }
      while not eolf and mess.help_index=help_index do
         begin
            clear_line(h,l);
            write(mess.line); l:=l+1;
            readf(helps,mess,eolf);
         end;
   end;

begin
{$NULLBODY}
end.
```

Users Module

```
{$WIDELIST}
{*****************************************************************}
{* Users Module - Maintains the List of Users for the Command    *}
{*                Processor                                      *}
{*****************************************************************}
program user_file;
const max_str = 30;      { Maximum length of a string }
    user_file = 3;
type
char_string = packed array[1..max_str] of char;
  whichfile = 1..10;
      links = record
                next, prev : integer;
              end;

{ List of Users File Type Definitions }
{ All the information maintained about a user }
   user_entry = record
                  link : links;
                  name : char_string;
                    id : integer;
              password : char_string;
               end;

{ Global Variable initialized by Read_Args routine. See
  the Argument Module for details. }
common next_user : integer;

{ File Access Declarations }
{ See the File Access Module for details on the following }
procedure readf(file_num : whichfile; var buffer : user_entry;
                var eolf : boolean); external;
procedure insertf(file_num : whichfile ; var rec_num : integer;
                  var buffer : user_entry); external;
procedure resetf(file_num : whichfile); external;
function roomf(file_num : whichfile) : integer; external;

{*****************************************************************}
{* Function : Lookup                                             *}
{* Parameters : User_Name, Password, ID                          *}
{* Entry Conditions : User_Name contains the name of the user    *}
{* as he typed it in.                                            *}
{* Process : User_Name is looked up in the User_File             *}
{* Exit Conditions : If the name is found, then Password and     *}
{* ID are set to the matching entries in the file, and the       *}
{* function returns a TRUE value.  If the name isn't in the     *}
{* list, the function returns a FALSE value.                     *}
{*****************************************************************}
function lookup(user_name : char_string;
                var password : char_string;
                var id : integer) : boolean;
   var user_info : user_entry;     { File entry for comparison }
```

Users Module

```
            eolf : boolean;          { End of List of Users }
   begin
      lookup:=false;    { Haven't found him yet }
      resetf(user_file);
      repeat
         readf(user_file,user_info,eolf);
         if user_info.name=user_name
            then begin  { This user IS present in the file }
                    password:=user_info.password;
                    id:=user_info.id;
                    lookup:=true { We found him }
                 end;
      until eolf or user_info.name=user_name;
   end;

{*****************************************************************}
{* Function : NewUserOK                                          *}
{* Entry Conditions : An Inquiry is being made to see if there*}
{* is room in the List of Users for another entry.               *}
{* Process : The RoomF function is called to see if there is  *}
{* any space available.                                          *}
{* Exit Conditions : If there is at least one entry available,*}
{* then the function will return a TRUE value.  Otherwise, the*}
{* result will be FALSE.                                         *}
{*****************************************************************}
function newuserok : boolean;
   begin
      if roomf(user_file)>0
      then newuserok:=true
      else newuserok:=false;
   end;
{*****************************************************************}
{* Function : Add_User                                           *}
{* Parameters : Name, Password                                   *}
{* Global Variables : Next_User                                  *}
{* Result : User_ID                                              *}
{* Entry Conditions : Name and Password are to be added to the*}
{* List of Users.                                                *}
{* Process : The Next_User id will be assigned to this Name   *}
{* Password pair and will be inserted into the List of Users. *}
{* Next_User will be incremented.                                *}
{* Exit Conditions : The function result will be the id       *}
{* assigned to the user.                                         *}
{*****************************************************************}
function add_user(name,password : char_string) : integer;
   access next_user;
   var user_info : user_entry; { File entry buffer }
               r : integer;    { The physical record number of
                                 the new entry - not used here }
   begin  { add user }
      resetf(user_file);
   { Set up the Buffer }
```

```
      user_info.name:=name;
      user_info.password:=password;
   { Use the Next_User id and update it }
      user_info.id:=next_user; next_user:=next_user+1;
      add_user:=user_info.id;
      insertf(user_file,r,user_info);
   end; { add user }

begin
{$NULLBODY}
end.
```

Projects Module

```
{$WIDELIST}
{*****************************************************************}
{* Projects Module - Maintains the List of Projects for the    *}
{*                   Command Processor                           *}
{*****************************************************************}
program proj_list;
const max_str = 30;    { Maximum length of a string }
      max_item = 20;   { Maximum length of a menu item }
     { See File Access Module for description of Files Array }
      projects = 4;    { Slot number in Files Array for Projects}
      g_dir = 7;       { Slot number of Global Directory }

type char_string = packed array[1..max_str] of char;
     menu_string = packed array[1..max_item] of char;
       whichfile = 1..10;
           links = record
                     next, prev : integer;
                   end;
            byte = 0..255;
{ List of Projects Type Definitions }
   state = (select_board, select_component, sel_done,
            connections, conn_done, placement, place_done,
            routing, route_done);
   proj_entry = packed record
                   link : links;
                   id   : integer;     { Project ID number }
                   name : char_string; { Project Name }
                   desc : char_string; { Project Description }
             completion : state;       { State of Completion }
                   user : integer;     { Owner ID number }
                end;

{ Global Variable - Initialized by Read_Args routine in
  Argument Module }
common next_proj : integer;  { Next project id to be assigned. }

{ File Access Declarations }
{ See File Access Module for details on the following }
procedure readf(file_num : whichfile; var buffer : proj_entry;
                var eolf : boolean); external;
procedure writef(file_num : whichfile; var buffer : proj_entry;
                var writeok : boolean); external;
procedure insertf(file_num : whichfile; var rec_num : integer;
                 var buffer : proj_entry); external;
procedure deletef(file_num : whichfile); external;
procedure resetf(file_num : whichfile); external;
function roomf(file_num : whichfile) : integer; external;

{ List building declarations }
{ See Menu Module for details on the following }
procedure init_list; external;
procedure build_list(item_text:menu_string;item_code:integer);
```

```
                        external;
function select_list : integer; external;

{ Returns number of Files Per Project - See Global Directory }
function fpp : integer; external;

{*****************************************************************}
{* Function : NewProjOK - Is there room for another project?  *}
{* Exit Conditions : If there is room in both the Global      *}
{* Directory and the List of Projects, then NewProjOK will be *}
{* TRUE.  Otherwise, FALSE will be returned.                  *}
{*****************************************************************}
function newprojok : boolean;
   begin
      if roomf(projects)>=1 and roomf(g_dir)>=fpp
      then newprojok:=true
      else newprojok:=false;
   end;

{*****************************************************************}
{* Procedure : Update_Proj_List - Update List of Projects     *}
{* Parameters : ID, Completion                                *}
{* Entry Conditions : ID identifies the project in the List   *}
{* of Projects to be updated, and Completion is the new value *}
{* for the state of the project.                              *}
{* Process : The List of Projects will be scanned for project *}
{* ID.  If found, its state of completion will be updated to  *}
{* Completion.  Nothing is done if ID is not found.           *}
{*****************************************************************}
procedure update_proj_list(id : integer; completion : state);
   var updated, eolf, writeok : boolean;
       proj_data : proj_entry;
   begin
      resetf(projects); updated:=false;
      repeat
         readf(projects,proj_data,eolf);
         if (proj_data.id=id)
            then begin  { We found it! }
                    proj_data.completion:=completion;
                    writef(projects,proj_data,writeok);
                    updated:=true;
                 end;
      until eolf or updated;
   end;

{*****************************************************************}
{* Function : Select_Project                                  *}
{* Parameters : User                                          *}
{* Result : One of the user's project IDs                     *}
{* Entry Conditions : User identifies whose projects to look  *}
{* up in the List of Projects.                                *}
{* Process : All of User's projects are looked up in the List *}
```

Projects Module

```
{* of Projects and the following information is inserted into *}
{* a list - project Name and ID number.  The Names will be    *}
{* displayed in a menu and the user will be asked to select   *}
{* one.                                                       *}
{* Exit Conditions : The ID number corresponding to the user's*}
{* selection will be the function result.  If the User has no *}
{* projects in the List of Projects, then a value of Zero will*}
{* be returned.                                               *}
{************************************************************}
function select_project(user : integer) : integer;
    var item_text : menu_string;
                c : 1..max_item;
        proj_data : proj_entry;
     gotone, eolf : boolean;
    begin
       gotone:=false;
       resetf(projects); init_list;
       repeat
          readf(projects,proj_data,eolf);
          if (proj_data.user=user) and not eolf
          then begin
                  gotone:=true;
                  for c:=1 to max_item do
                     item_text[c]:=proj_data.name[c];
                  build_list(item_text,proj_data.id);
               end;
       until eolf;
       if gotone
       then select_project:=select_list
       else select_project:=0;
    end;

{************************************************************}
{* Procedure : New_Project - Add a new project to the List    *}
{* Parameters : Name, Desc, User, ID                          *}
{* Entry Conditions : Name, Desc(ription), and User define the*}
{* new project to be added to the List of Projects.           *}
{* Exit Conditions : ID is the Project ID assigned to the     *}
{* project.                                                   *}
{************************************************************}
procedure new_project(var name, desc : char_string;
                      user : integer; var id : integer);
   access next_proj; { Global Variable  - Initialized by
                       Read_Header routine in Argument Module }
   var proj_data : proj_entry;
       r : integer;
   begin
      resetf(projects);
     { Load the Buffer }
      proj_data.name:=name;
      proj_data.desc:=desc;
      proj_data.user:=user;
```

```
      proj_data.completion:=select_board;
     { Use the next sequential ID and update it }
      proj_data.id:=next_proj; next_proj:=next_proj+1;
     { Return the new ID the the Caller }
      id:=proj_data.id;
      insertf(projects,r,proj_data);
   end; { new project }

{*****************************************************************}
{* Procedure : Free_Project - Remove a project from the List  *}
{* Parameters : Project_ID                                       *}
{* Entry Conditions : Project_ID identifies the project to be *}
{* removed from the List of Projects.                            *}
{* Process : List of Projects will be scanned for Project_ID. *}
{* If it is found, it will be deleted.  Nothing happens if the*}
{* project isn't in the list.                                    *}
{*****************************************************************}
procedure free_project(project_id : integer);
   var eolf : boolean;
       proj_data : proj_entry;
   begin
      if project_id<>0
      then begin
              resetf(projects);
              repeat
                 readf(projects,proj_data,eolf);
                 if proj_data.id=project_id
                 then deletef(projects);
              until eolf or (proj_data.id=project_id);
           end;
   end;

{*****************************************************************}
{* Function : Get_State - What is the state of the Project?   *}
{* Parameters : ID                                               *}
{* Result : The current State of Completion of the project    *}
{* Entry Conditions : ID identifies which project to look up  *}
{*****************************************************************}
function get_state(id : integer) : state;
   var eolf : boolean;
       proj_data : proj_entry;
   begin
      resetf(projects);
      repeat
         readf(projects,proj_data,eolf);
         if (proj_data.id=id)
         then get_state:=proj_data.completion;
      until eolf or (proj_data.id=id);
   end;

{*****************************************************************}
{* Procedure : Get_Name - What is the name of the project?    *}
```

Projects Module

```
{* Parameters : ID, Name                                              *}
{* Entry Conditions : ID identifies the project to look up           *}
{* Exit Conditions : Name will contain the name of the project       *}
{* if ID exists.  If ID isn't in the List of Projects, name          *}
{* will be set to all spaces.                                        *}
{*******************************************************************}
procedure get_name(id : integer; var name : char_string);
   var eolf : boolean;
       proj_data : proj_entry;
   begin
      name:='                                    ';
      resetf(projects);
      repeat
         readf(projects,proj_data,eolf);
         if (proj_data.id=id) and not eolf
         then name:=proj_data.name;
      until eolf or (proj_data.id=id);
   end;

begin
{$NULLBODY}
end.
```

Argument Module

```
{$WIDELIST}
{*****************************************************************}
{* Argument Module - Responsible for passing arguments between*}
{* the Command Processor and the other layout modules.        *}
{*****************************************************************}
program args;
const args = 5;
      g_dir = 7;
type
      wf = 1..10;
{ Argument File Definitions }
   state = (select_board, select_component, sel_done,
            connections, conn_done, placement, place_done,
            routing, route_done);
 modules = (cp, selecter, connecter, placer, router, os);
{ Arg_Header contains information the CP needs to know. }
  arg_header = record
                 project_id,          { Current Project }
                 user_id,             { Current User }
                 error_code : integer;{ Current Error }
                 completion : state;  { State of Completion }
                 module : modules;    { Last module executed }
               end;
{ Saved_State keeps track of the next IDs to assign }
  saved_state = record
                  next_id,
                  next_user,
                  next_proj : integer;
                  proj_info : arg_header;
                end;

    filespec = array[1..12] of char;
    drive_id = 0..1;
       links = record
                 next, prev : integer;
               end;
       where = record
                  file_name : filespec;
                  linked    : boolean;
                  drive     : drive_id;
                  disk_id,
                  recs_avail,
                  rec_len,
                  first,
                  free      : integer;
               end;
   arg_entry = record
                 link : links;
                 case boolean of
                 false:(header : saved_state);
                  true:(file_entry : record
                                       file_num : wf;
```

Argument Module

```
                                        file_loc : where;
                                       end);
            end;

{ Global Variables }
common next_proj, next_user, next_id : integer;

{ See the LinkFile Module for details on the following }
procedure readf(f : wf; var buffer:arg_entry; var eolf:boolean);
             external;
procedure writef(f : wf; var buffer:arg_entry;
              var writeok:boolean); external;
procedure resetf(f : wf); external;
procedure insertf(f : wf; var rec_num : integer;
               var buffer : arg_entry); external;
procedure deletef(f : wf); external;

{ See the Global Directory Module for details on the following }
procedure update_file(module : modules; project : integer;
                  file_num : wf; file_loc : where); external;
function get_loc(module : modules; project : integer;
   var file_num : wf; var file_loc : where) : boolean; external;

{*****************************************************************}
{* Procedure : Read_Args - Read in arguments passed              *}
{* Parameters : Info                                             *}
{* Entry Conditions : The Argument File contains the current     *}
{* state of the system.                                          *}
{* Process : The current state is sent back to the Command       *}
{* Processor in Info.  If the last module executed updated       *}
{* any files, then the Global Directory has to be updated.       *}
{*****************************************************************}
procedure read_args(var info : arg_header);
   access next_proj, next_user, next_id;
   var eolf : boolean;
       buffer : arg_entry;
   begin
      resetf(args);
      readf(args,buffer,eolf);
      info:=buffer.header.proj_info;;
      next_proj:=buffer.header.next_proj;
      next_user:=buffer.header.next_user;
      next_id:=buffer.header.next_id;
      repeat  { Read current state of files }
         readf(args,buffer,eolf);
         if not eolf
         then begin
                deletef(args);
                with buffer.file_entry do
              { Update the Global Directory to match }
                 update_file(info.module,info.project_id,
                 file_num,file_loc);
```

```
            end;
      until eolf;
   end;

{*****************************************************************}
{* Procedure : Update_Header - Save current state                *}
{* Parameters : Info                                             *}
{* Entry Conditions : Info is the current state of the system    *}
{* to be save in the Argument File.                              *}
{*****************************************************************}
procedure update_header(info : arg_header);
   access next_proj,next_user,next_id;
   var eolf, writeok : boolean;
       buffer : arg_entry;
   begin
      resetf(args);
      readf(args,buffer,eolf);
      buffer.header.next_proj:=next_proj;
      buffer.header.next_user:=next_user;
      buffer.header.next_id:=next_id;
      buffer.header.proj_info:=info;
      writef(args,buffer,writeok);
   end;

{*****************************************************************}
{* Procedure : Load_Args - Load Argument File with files         *}
{* Parameters : Module, Project                                  *}
{* Entry Conditions : Module and Project identify which files    *}
{* are to be loaded.                                             *}
{* Process : The Global Directory is scanned for files with      *}
{* matching Module and Project parameters.  When found, their    *}
{* locations are inserted into the Argument File.                *}
{*****************************************************************}
procedure load_args(module : modules; project : integer);
   var buffer : arg_entry;
            r : integer;
         eolf : boolean;
   begin
      { Reset the Directory and the Argument files, and skip the
        Argument File Header record. }
      resetf(g_dir); resetf(args); readf(args,buffer,eolf);
      repeat { Scan the Global Directory }
        with buffer.file_entry do
           eolf:=get_loc(module,project,file_num,file_loc);
        if not eolf
        then insertf(args,r,buffer);
      until eolf;
   end;

begin
{$NULLBODY}
end.
```

DiskList Module

```
{$WIDELIST}
{*****************************************************************}
{* DiskList Module - Keeps track of allocated and available    *}
{*                   Diskette space.                             *}
{* A record is maintained for each diskette in the system.     *}
{* Each diskette may be either assigned to a specific project, *}
{* or assigned to a user.  If assigned to a user, then it is   *}
{* avaialable for assignment to a project created by that user *}
{* only.                                                       *}
{*****************************************************************}
program disk_list;
const max_space = 133; { Maximum space available in Kbytes }
      disks = 6;
type
    wf = 1..10;
    links = record
              next, prev : integer;
            end;
    disk_assgn = record
                     link : links;
                  disk_id,            { Which disk is it }
               free_space : integer;  { How much room is
                                        available }
              case assigned : boolean of
         true : (project_id : integer);
          false : (user_id : integer);
         end;

{ Global Variable initialized by Read_Args - See Argument
  Module for details }
common next_id : integer;

{ File Access Declarations }
{ See the LinkFile Module for details on the following }
procedure readf(f : wf; var buffer : disk_assgn;
                var eolf : boolean); external;
procedure writef(f : wf; var buffer : disk_assgn;
                var writeok : boolean); external;
procedure insertf(f : wf; var rec_num : integer;
                 var buffer : disk_assgn); external;
procedure resetf(f : wf); external;

{ See the DiskID Module for details on the following }
procedure new_disk(disk_id : integer) ;external;

{*****************************************************************}
{* Function : Get_Disk_ID - Assign a project file to a disk    *}
{* Parameters : User, Project, Size                            *}
{* Result : Disk_ID assigned to file                           *}
{* Entry Conditions : User identifies the owner of the file,   *}
{* Project identifies the project to which the file belongs,   *}
{* and Size is the number of KBytes required by the file.      *}
```

DiskList Module

```
{* Process : The Disks File is scanned for a diskette already *)
{* assigned to Project with enough free space on it.  If one  *)
{* can't be found, then the file is scanned for a diskette    *)
{* belonging to User.  If one is found, it is assigned to pro-*)
{* ject.  If the User has no free disks, then a new diskette  *)
{* will be added to the system and assigned to the project.   *)
{************************************************************}
function get_disk_id(user, project, size : integer) : integer;
   access next_id;
   var done, eolf, writeok : boolean;
       disk_data : disk_assgn;
       r : integer;
   begin
      resetf(disks); done:=false;
      repeat
        { Look for a diskette already assigned to Project with
          enough free space on it for the file }
        readf(disks,disk_data,eolf);
        if (disk_data.assigned
            and disk_data.project_id=project
            and disk_data.free_space>=size)
           then with disk_data do
                begin
                  { Found One! }
                  get_disk_id:=disk_id;
                  free_space:=free_space-size;
                  writef(disks,disk_data,writeok);
                  done:=true;
                end;
      until eolf or done;
      if not done
      then begin
             resetf(disks);
             { Next, look for a free disk assigned to User }
             repeat
                readf(disks,disk_data,eolf);
                if (not disk_data.assigned and
                    disk_data.user_id=user)
                then with disk_data do
                     begin
                     { Assign it to Project }
                        get_disk_id:=disk_id;
                        free_space:=max_space-size;
                        assigned:=true;
                        project_id:=project;
                        writef(disks,disk_data,writeok);
                        done:=true;
                     end;
             until eolf or done;
           end;
      if not done
         then with disk_data do
```

```
                begin
                  { Have to make a new diskette available }
                   disk_id:=next_id; next_id:=next_id+1;
                   get_disk_id:=disk_id;
                   assigned:=true;
                   project_id:=project;
                   insertf(disks,r,disk_data);
                  { Label and identify the new diskette }
                   new_disk(disk_id);
                end;
   end;

{*****************************************************************}
{* Procedure : Free_Disk - Deallocate diskette space             *}
{* Parameters : User, Project                                    *}
{* Entry Conditions : Project identifies the project which is    *}
{* is no longer active, and User identifies who the released     *}
{* space will be assigned to.                                    *}
{*****************************************************************}
procedure free_disk(user,project : integer);
   var eolf, writeok : boolean;
       disk_data : disk_assgn;
   begin
      resetf(disks);
      repeat
         readf(disks,disk_data,eolf);
         if disk_data.assigned and
            disk_data.project_id=project
         then begin
                 disk_data.assigned:=false;
                 disk_data.user_id:=user;
                 disk_data.free_space:=max_space;
                 writef(disks,disk_data,writeok);
              end;
      until eolf;
   end;

begin
{$NULLBODY}
end.
```

Global Directory Module

```
{$WIDELIST}
{*****************************************************************}
{* Global Directory Module - Maintains the directory of all    *}
{* files that the system has knowledge of.                     *}
{*****************************************************************}
program global_directory;
const g_dir = 7;
      dump = 8; { Temporary slot used for files }
type
 whichfile = 1..10;
 links     = record
               next, prev : integer;
             end;
  { Global Directory Type Definitions }
  { Which Module(s) can access the file? }
   modules = (cp,selecter,connecter,placer,router,os);
  { System Dependent File Specification }
  filespec = packed array[1..12] of char;
  { System Dependent Drive Identification }
  drive_id = 0..1;
  { Where is the file located ?}
     where = record
               file_name : filespec;  { Name of the File }
               linked    : boolean;
               drive     : drive_id;  { Drive in which to
                                        Mount Diskette }
                 disk_id,  { Which diskette the file is on }
                 rec_len,  { The record length (power of two)}
              recs_avail,  { Number of unused records in file}
                   first,  { Physical record number of first
                             logical record }
                    free : integer; { Physical record number of
                             free space list }
             end;
  gd_entry = record
                link      : links;
                module_id : packed array[modules] of boolean;
                            { True value means the module has
                              access to the file }
                 file_num : whichfile; { Files Array Index }
               project_id,  { Project to which file belongs. A
                              value of Zero indicates a file
                              that is used for all projects }
                 how_many : integer; { How Many records have
                                       been allocated }
                 file_loc : where;
             end;

{ See the File Access Module for details on the following }
procedure statef(file_num:whichfile; var first,free : integer);
    external;
function roomf(file_num : whichfile) : integer;
```

```
     external;
procedure resetf(file_num : whichfile);
     external;
procedure initf(file_num : whichfile; file_loc : where);
     external;
procedure closef(file_num : whichfile);
     external;
procedure close_all(drive : drive_id);
     external;
procedure readf(file_num : whichfile; var buffer : gd_entry;
                var eolf : boolean);
     external;
procedure writef(file_num : whichfile; var buffer : gd_entry;
                 var writeok : boolean);
     external;
procedure insertf(file_num : whichfile; var rec_num : integer;
                  var buffer : gd_entry);
     external;
procedure deletef(file_num : whichfile);
     external;
procedure createf(file_num : whichfile; how_many : integer);
     external;
procedure erasef(file_num : whichfile);
     external;
procedure run_file(file_name : filespec; drive : drive_id;
                   disk_id : integer);
     external;
procedure init_files;
     external;

{ See the Menu Module for details on the following }
procedure init_menu;
     external;

{ See the Disk List Module for details on the following }
function get_disk_id(user, project, space : integer):integer;
     external;

{*****************************************************************}
{* Procedure : New_files                                         *}
{* Parameters : User, Project                                    *}
{* Entry Conditions : User and Project identify a new project    *}
{* that the user is creating.                                    *}
{* Process : The Global Directory is scanned for Template        *}
{* Entries.  A Template Entry is identified by a value of Zero   *}
{* in the Disk_ID field.  For each Template Entry found, a new   *}
{* file is created and an entry is made in the Global            *}
{* Directory.                                                    *}
{*****************************************************************}
procedure new_files(user, project : integer);
   var new_gde : gd_entry;  { Buffer for New Entries }
    r, maxsize : integer;   { R is the physical record number of
```

```
                                newly inserted entries - not used
                                here.  Maxsize is the number of
                                Kbytes required by the file. }
            eolf : boolean;    { End of Directory indicator }
   begin
      resetf(g_dir);
      repeat
         readf(g_dir,new_gde,eolf);
         if new_gde.file_loc.disk_id=0
         then begin  { Located a Template File }
                 with new_gde.file_loc do
                 begin
                    maxsize:=
                    ((rec_len * new_gde.how_many) div 1024) + 1;
                 { Allocate Diskette space for the file }
                    disk_id:=get_disk_id(user,project,maxsize);
                 end;
                { Assign the file to the Project }
                 new_gde.project_id:=project;
                { Create the new file }
                 initf(dump,new_gde.file_loc);
                 createf(dump,new_gde.how_many);
                 closef(dump);
                { Add it to the Directory }
                 insertf(g_dir,r,new_gde);
              end;
      until eolf;
   end;

{*****************************************************************}
{* Function : FPP - Files Per Project                            *}
{* Result : The number of files created for every project        *}
{*****************************************************************}
function fpp : integer;
   begin
      fpp:=2 { Value depends on current system configuration }
   end;

{*****************************************************************}
{* Procedure : Kill_GDE                                          *}
{* Parameters : Project_ID                                       *}
{* Entry Conditions : Project_ID identifies the project whose    *}
{* files are to be removed from the Directory.                   *}
{* Process : The Directory is scanned looking for Entries with   *}
{* a Project_ID that matches the parameter. When found, those    *}
{* Directory Entries are deleted and the corresponding file is   *}
{* Purged from the system.                                       *}
{*****************************************************************}
procedure kill_gde(project_id : integer);
   var gde : gd_entry; { Directory Entry Buffer }
      eolf : boolean;  { End of Directory flag }
   begin { kill }
```

```
      if project_id<>0
         then begin
                 resetf(g_dir);
                 repeat
                    readf(g_dir,gde,eolf);
                    if gde.project_id=project_id
                    then begin  { Found one! }
                            deletef(g_dir);
                            initf(dump,gde.file_loc);
                            erasef(dump);
                         end;
                 until eolf;
              end;
   end; { kill }

{*****************************************************************}
{* Procedure : UpdateGD   Update Global Directory                *}
{* Process : Before the Command Processor Finishes execution,    *}
{* the new status of all of the files must be recorded in the    *}
{* Directory to keep it current.  As records are added to or     *}
{* deleted from the Command Processor Files, the First, Free,    *}
{* and Recs_Avail parameters are subject to change.              *}
{*****************************************************************}
procedure updategd;  { Must be called before CP terminates!!!}
   var eolf, writeok : boolean;
                 gde : gd_entry;
first, free, recs_avail : integer; { Parameters to be updated }
   begin
      resetf(g_dir);
      repeat
         readf(g_dir,gde,eolf);
         if not eolf and gde.module_id[cp]
            then begin
                    { Get the current state of the file }
                    statef(gde.file_num,first,free);
                    gde.file_loc.first:=first;
                    gde.file_loc.free:=free;
                    { Get the number of available records }
                    gde.file_loc.recs_avail:=roomf(gde.file_num);
                    writef(g_dir,gde,writeok);
                 end;
      until eolf;
     { Make sure all the files get updated in the OS directory }
      close_all(0);
      close_all(1);
   end;

{*****************************************************************}
{* Procedure : Update_File - Update a file's  GD Entry           *}
{* Parameters : File_Num, Project, File_Loc                      *}
{* Entry Conditions : File_Num and Project identifies the file*}
{* whose Global Directory Entry must be updated.                 *}
```

Global Directory Module

```
{* Process : The file is located in the Directory and the     *}
{* entry is updated with the information in File_Loc.         *}
{*************************************************************}
procedure update_file(module : modules; project : integer;
                      file_num : whichfile;  file_loc : where);
   var eolf, writeok : boolean;
       gde : gd_entry;
   begin
      resetf(g_dir); writeok:=false;
      repeat
         readf(g_dir,gde,eolf);
         if ((gde.file_num=file_num) and
             (gde.project_id=project) and
             (gde.module_id[module]))
         then begin  { This is the Entry to be Updated }
                 gde.file_loc:=file_loc;
                 writef(g_dir,gde,writeok);
              end;
      until eolf or writeok;
   end;

{*************************************************************}
{* Function : Get_Loc - Locate files for other Modules       *}
{* Parameters : Module, Project, File_Num, File_Loc          *}
{* Result : EOLF condition                                   *}
{* Entry Conditions : Module and Project identify which files *}
{* are to be looked for in the Directory.                    *}
{* Process : The Directory is scanned from the current       *}
{* looking for files that belong to the Module and Project.  *}
{* If one is found, its number and location are returned to  *}
{* the caller.  This function should be called repeatedly    *}
{* until an End of File condition is signalled.              *}
{* Exit Conditions : File_Num and File_Loc are returned, and *}
{* the function returns a FALSE value, if a file was located *}
{* that belonged to the module and project.  If the End of   *}
{* file is reached, then the function returns a TRUE value.  *}
{*************************************************************}
function get_loc(module : modules; project : integer;
    var file_num : whichfile; var file_loc : where) : boolean;
   var eolf : boolean;
       gde : gd_entry;
   begin
      repeat
         readf(g_dir,gde,eolf);
      until eolf or ((gde.module_id[module]) and
           ((gde.project_id=project) or (gde.project_id=0)));
      file_num:=gde.file_num;
      file_loc:=gde.file_loc;
      get_loc:=eolf;
   end;

{*************************************************************}
```

Global Directory Module

```
{* Procedure : Init_All  Initialize Everything!                *}
{* Process : After some Miscellaneous initialization routines *}
{* are called, the Global Directory is Initialized.  Then all *}
{* of the Command Processor Files are looked up in the        *}
{* Directory, and they are initialized.  No access may be made*}
{* to a file until it is initialized!                         *}
{************************************************************}
procedure init_all;
   var file_loc : where;
           eolf : boolean;
            gde : gd_entry;
   begin
      init_files; { Set up the file array with all files closed}
      init_menu;  { Set the menu pointers to nil }
      { The global directory must be initialized }
      with file_loc do
      begin
        { You have to know where to find the Directory }
         disk_id:=1;
         file_name:='GLOBAL/DIR:0';
         drive:=0;
         rec_len:=64;
         first:=0;              {This should'nt change!!!}
      end;
      initf(g_dir,file_loc);
     { Free and Recs_Avail parameters still not set }
      resetf(g_dir);
     { Look up Directory entry in the Directory to set them }
      repeat
         readf(g_dir,gde,eolf);
      until gde.file_num=g_dir;
      closef(g_dir); initf(g_dir,gde.file_loc);
      { Now the rest of the files can be located }
      resetf(g_dir);
      repeat
        readf(g_dir,gde,eolf);
        if (not eolf and gde.file_num<>g_dir and
           gde.module_id[cp] and gde.project_id=0)
          then initf(gde.file_num,gde.file_loc);
     until eolf;
   end;

{************************************************************}
{* Procedure : Exec - Execute an Operating System Program     *}
{* Parameters : Module                                        *}
{* Entry Conditions : Module identifies the Program to be     *}
{* executed.  The Command Processor will no longer be in      *}
{* control.                                                   *}
{************************************************************}
procedure exec(module : modules);
   var gde : gd_entry;
       eolf : boolean;
```

Global Directory Module

```
   begin
     { To execute OS, all that is necessary is to return from
       this procedure, after which the program will end! }
      if module<>os
     { Otherwise, the proper module must be executed. }
      then begin
              resetf(g_dir);
              repeat
                 readf(g_dir,gde,eolf);
                 if gde.module_id[module] and gde.module_id[os]
                 then with gde.file_loc do
                      run_file(file_name,drive,disk_id);
              until eolf;
           end;
   end; { If you get to here, then the OS will take control. }

begin
{$NULLBODY}
end.
```

Command Processor Module

```
{$WIDELIST}
{*****************************************************************}
{*  Command Processor Module                                     *}
{*****************************************************************}
{*   The following files must exist on the master CP disk        *}
{*   (Disk number 1):                                            *}
{*   GLOBAL/DIR - describes the parameters of the rest           *}
{*                of the files                                   *}
{*   MENUS/CP   - Contains menus 1 and 2                         *}
{*                (See Select_Option and Get_Next_Module         *}
{*                for menu contents)                             *}
{*   HELP/CP    - Contains all of the system Help                *}
{*                messages.                                      *}
{*   USERS/CP   - A list of all Users and their ID's             *}
{*   PROJECT/CP - A list of all Projects and their ID's          *}
{*   DISKS/CP   - Contains Diskette space allocation             *}
{*   ARGS/CP    - The argument file is the means of              *}
{*                communication between the system               *}
{*                modules.  It also contains "Current            *}
{*                State" information that must be saved          *}
{*                between Command Processor sessions.            *}
{*                In particular, the Command_Processor           *}
{*                Global Variables are initialized with          *}
{*                the following information:                     *}
{*   project_id - Number of Last project accessed                *}
{*   user_id    - Identity of current user                       *}
{*   error_code - Error conditions of other modules              *}
{*   completion - State of completion of project_id              *}
{*   Exit Conditions : The Global Directory and Argument         *}
{*   Files are updated, and either the program is exited, or     *}
{*   the next layout module is invoked.                          *}
{*****************************************************************}
program command_processor;
const max_prompt = 30;      { Maximum length of a prompt }
      max_num = 10;         { Maximum length of a number }
      max_str = 30;         { Maximum length of a string }

type
{ Query Type Definitions }
            unit = (mils, inches, mm, scalar);
    prompt_string = packed array[1..max_prompt] of char;
            p_len = 1..max_prompt;
         position = 1..max_str;
      string_case = (upper, lower, none);
      char_string = packed array[1..max_str] of char;
            yesno = (yes, no, i_dunno);
            boxes = (q, m, h);

{ Global Directory Type Definitions }
   modules = (cp,selecter,connecter,placer,router,os,undefined);
             { A list of the possible values for the next module
               to be executed }
```

Command Processor Module

```
{ List of Projects Type Definitions }
   state = (select_board, select_component, sel_done,
            connections, conn_done, placement, place_done,
            routing, route_done);
           { A list of the possible values for the state of
             completion of a project }

{ Argument File Type Definitions }
{ Arg_Header defines the information the Command_Processor reads
  from the ARGS/CP file to initialize the global variables.}
  arg_header = record
                  project_id,
                  user_id,
                  error_code : integer;
                  completion : state;
                  module : modules;
               end;

var { Global Variables }
     current : arg_header;
       valid : boolean;    { Indicates the validity of a user }

{ Query Declarations }
{ See the Query Modules for an explanation of these routines }
procedure querynum(prompt : prompt_string;
            prompt_length : p_len;
                 min, max : integer;
               var answer : integer;
                    units : unit;
               help_index : integer); external;
procedure querystr(prompt : prompt_string;
            prompt_length : p_len;
                 min, max : position;
               var answer : char_string;
        var string_length : position;
                make_case : string_case;
               help_index : integer); external;
procedure query_yn(prompt : prompt_string;
            prompt_length : p_len;
               var answer : yesno;
               help_index : integer); external;

{ Screen I/O Declarations }
{ See the Terminal I/O Module for an explanation of these
  routines. }
procedure goto_box(box : boxes; x,y : integer); external;
procedure clear_line(box : boxes; y : integer); external;
procedure clear_box(box : boxes); external;
function get_count(box : boxes) : integer; external;
procedure input_error(err_msg : char_string); external;
```

Command Processor Module

```
{ See the Global Directory Module for an explanation of this
  routine. }
procedure init_all; external;

{*****************************************************************}
{* Procedure Name : Resolve                                      *}
{* Parameters : Error_Code                                       *}
{* Entry Conditions : Error_Code identifies a particular error*}
{* condition that another module was unable to handle.           *}
{* Exit Conditions : If possible, the error condition will be *}
{* corrected and the error_code will be reset to zero.           *}
{* In the current implementation, this is a dummy routine, but*}
{* it is provided for future expansion.                          *}
{*****************************************************************}
procedure resolve(var error_code : integer);
   begin
      write('***** ERROR ',error_code:3,' *****');
      error_code:=0
   end;

{*****************************************************************}
{* Procedure : Get_Identity                                      *}
{* Parameters : ID, Valid                                        *}
{* Entry Conditions : The identity of the user is unknown.       *}
{* Exit Conditions : ID will identify the user if he exists.     *}
{* Valid is a flag that will indicate if the user exists.        *}
{*****************************************************************}
procedure get_identity(var id:integer; var valid:boolean);
  const no_password = '                                ';
   var      name,                  { The name of the user }
        password : char_string;    { His Password }
        name_len : position;       { The number of characters
                                     in the user's name }
           found : boolean;        { A Flag indicating whether
                                     or not the user already
                                     exists }
          answer : yesno;          { Will indicate if the user
                                     typed his name correctly }

{ See the Users Module for details on the next three routines }
   function lookup(user_name : char_string;
                   var password : char_string;
                   var id : integer) : boolean; external;
   function add_user(user_name,password:char_string):integer;
                     external;
   function newuserok : boolean; external;

{*****************************************************************}
{* Procedure : Create_User                                       *}
{* Parameters : Name                                             *}
{* Outer Level Variables : Password, ID, Valid                   *}
{* Entry Conditions : Name was not found in the List of          *}
```

Command Processor Module

```
{* Current Users.                                                    *}
{* Exit Conditions : If there is room for another user in the *}
{* List of Current Users and the operator indicated a desire  *}
{* to become a user, then ID and Password will be updated and *}
{* Vaild will be set to true.  Otherwise, Valid will be false.*}
{*****************************************************************}
   procedure create_user(name : char_string);
      var answer : yesno;          {Will indicate whether or not
                                    the operator wishes to become
                                    a user }

{*****************************************************************}
{* Procedure : Get_Password                                       *}
{* Parameters : Password                                          *}
{* Entry Conditions : A new user must be assigned a Password  *}
{* Exit Conditions : If the user indicated that he didn't want*}
{* a password, then Password will be set to all spaces.       *}
{* Otherwise, Password will be set to whatever character      *}
{* string he entered.                                             *}
{*****************************************************************}
      procedure get_password(var password : char_string);
         var answer : yesno;      { Will indicate whether or not
                                    a password is desired }
                len: position;  { The length of the password }
         begin
            query_yn('Do you want a Password?       ',23,
                     answer,4);
            if answer=yes then
               querystr('What will your Password be?   ',27,
                        5,20,password,len,none,5)
               else password:=no_password;
         end;  { get_password }

      begin { create_user }
         if newuserok     { There IS room for another one }
         then begin
                 query_yn('Do you wish to become a user? ',29,
                          answer,3);
                 if answer=yes
                    then begin { A New User!!!}
                            get_password(password);
                            id:=add_user(name,password);
                            valid:=true;
                         end
                    else valid:=false;
              end
         else input_error('No More Room for New Users!   ');
      end; { create_user }

{*****************************************************************}
{* Function : Verify_Identity                                     *}
{* Parameters : Password                                          *}
```

Command Processor Module

```
{* Entry Conditions : The users name was located in the List  *}
{* of Current Users, and Password contains the password that  *}
{* the user originally specified.                             *}
{* Exit Conditions : If the comparison string entered by the  *}
{* user matches his original Password, or if the original is  *}
{* all spaces, then the function returns a True result.       *}
{* Otherwise, a False value is returned.  The user get three  *}
{* tries to get the password right.                           *}
{************************************************************}
   function verify_identity(password : char_string) : boolean;
      var testword : char_string;  { The string to compare with
                                     the original Password }
               len : position;     { The number of characters in
                                     testword }
             tries : 0..3;         { Counts the number of
                                     attempts to match }
             valid : boolean;      { Indicates result of com-
                                     parison }
      begin
         if password=no_password
            then verify_identity:=true
            else begin
                    tries:=0; valid:=false;
                    repeat
                       querystr('Prove It!                        ',9,
                              5,20,testword,len,none,6);
                       if testword=password then valid:=true;
                       tries:=tries+1;
                    until valid or tries=3;
                    verify_identity:=valid;
                 end
      end; { verify_identity }

   begin { get_identity }
     repeat
        querystr('Who Are You ?                  ',13,
                 2,20,name,name_len,upper,1);
        found:=lookup(name,password,id); answer:=yes;
        if not found
           then begin
                   clear_line(q,-1);
                   write(name);
                   query_yn('Is your name correct?          ',21,
                            answer,2);
                   clear_line(q,-1);
                end;
     until answer=yes;
     if not found
        then create_user(name)
        else valid:=verify_identity(password);
   end; { get_identity }
```

Command Processor Module

```
{*****************************************************************}
{* Procedure : Select_Option                                     *}
{* Parameters : User_ID, Project_ID, Completion, Next_Module     *}
{* Entry Conditions : User_ID identifies the current user.       *}
{* Exit Conditions : Project_ID, Completion, and Next_Module     *}
{* will have been updated to reflect the current status of the*}
{* project selected by the user.                                 *}
{*****************************************************************}
procedure select_option(var user_id : integer;
                        var project_id : integer;
                        var completion : state;
                        var next_module : modules);

   var  name : char_string;    { Will contain the name of the
                                 current project }
        selection : integer;   { The Menu Selection Index }

{See the Projects Module for details on the following procedure}
   procedure update_proj_list(id : integer; completion : state);
                              external;

{See the Menu Module for details on the following function }
   function menu(num : integer) : integer; external;

{*****************************************************************}
{* Function : Get_Next_Module                                    *}
{* Parameters : Completion                                       *}
{* Entry Conditions : The user has selected a project and        *}
{* desires to work on it.  Completion identifies the current     *}
{* state of the selected project.                                *}
{* Exit Conditions : The function result is the Next Step in     *}
{* Layout Process that the user has selected.  A given step      *}
{* may be repeated as often as desired, but may only be per-     *}
{* formed if all of the previous steps have been completed.      *}
{*****************************************************************}
   function get_next_module(completion : state) : modules;
      var selection : integer;       { Will contain the selection
                                       from Menu 2 }
          answer : yesno;            { Will contain response to
                                       query on repeating a step}
          highest, next: modules;    { Will identify the highest
                                       allowable step and the
                                       user's selection }
      begin { get_next_module }
         case completion of
          select_board, select_component : highest:=selecter;
                   sel_done, connections : highest:=connecter;
                    conn_done, placement : highest:=placer;
         place_done, routing, route_done : highest:=router;
         end;
         repeat
            repeat
```

```
            selection:=menu(2);
            { 1 : Selecter
              2 : Connecter
              3 : Placer
              4 : Router }
            case selection of
            1 : next:=selecter;
            2 : next:=connecter;
            3 : next:=placer;
            4 : next:=router;
            end;
            if next>highest
            then input_error('You Can''t do that yet!        ');
         until next<=highest;
         if next<highest
            then query_yn('You want to redo this step?   ',27,
                          answer,16)
            else answer:=yes;
      until answer=yes;
      get_next_module:=next;
   end; { get_next_module }

{ See Projects Module for details on the following }
   function select_project(user_id : integer) : integer;
                           external;
   function get_state(id : integer) : state; external;
   procedure get_name(id:integer;var name:char_string);external;
   function newprojok : boolean; external;
   procedure new_project(var name, desc : char_string;
                         user : integer; var id : integer);
                         external;

{ See Global Directory Module for details on the following }
   procedure new_files(user, project : integer); external;

{*****************************************************************}
{* Procedure : Create_Project                                    *}
{* Parameters : User, ID, Name                                   *}
{* Entry Conditions : The user has decided to create another     *}
{* project.  User is the identification number of the current    *}
{* user to whom the new project will be assigned.                *}
{* Exit Conditions : If there is room for another project,       *}
{* then ID will contain the project identification number that   *}
{* was assigned to it, and Name will contain the user assigned   *}
{* name. If there is not room for another project to be          *}
{* defined, then ID will be zero.                                *}
{*****************************************************************}
   procedure create_project(user : integer; var id : integer;
                            var name : char_string);
      var desc : char_string; { User's description of project }
          ans_len : position; { Number of characters in name }
      begin
```

```
        if newprojok
        then begin  { There IS room for another project }
             querystr('What do you want to call it?  ',28,2,20,
                      name,ans_len,none,20);
             querystr('Give a brief description :    ',26,1,30,
                      desc,ans_len,none,21);
           { Add the project to List of Projects }
             new_project(name,desc,user,id);
           { Allocate Diskette space for the project }
             new_files(user,id);
             end
        else begin
             input_error('No More Room for Projects!    ');
             id:=0;
             end;
      end;

{ See the Global Directory Module for details on the following }
   procedure kill_gde(project : integer); external;

{ See the Projects Module for details on the following }
   procedure free_proj(project_id : integer); external;

{ See the Disk List Module for details on the following }
   procedure free_disk(user, project : integer); external;

{*****************************************************************}
{* Procedure : Kill_Project                                      *}
{* Parameters : User, Project                                    *}
{* Entry Conditions : The user has decided to destroy the        *}
{* currently selected project identified by Project.             *}
{* Exit Conditions : The project will have been removed from     *}
{* the List of Projects and the diskette space will have been    *}
{* de-allocated.  Also, Project will have been reset to zero     *}
{* to indicate that there is no longer a current project.        *}
{*****************************************************************}
   procedure kill_project(user:integer; var project:integer);
      begin
        { Remove the Project from the Directory }
         kill_gde(project);
        { Remove the project from the List of Projects }
         free_project(project);
        { De-allocate the Diskette space }
         free_disk(user,project);
        { Reset to No Current Project }
         project:=0;
      end;

   begin { Select Option }
      if project_id<>0
      then begin { There IS a currently selected project }
             { Make sure the List of Projects is Up to Date }
```

```
                update_proj_list(project_id,completion);
                get_name(project_id,name);
             end
        else name:='No Current Project Selected.  ';
        next_module:=undefined;
        repeat
           clear_line(q,3); write('Current Project :',name);
           selection:=menu(1);
           { 1 : Continue Project
             2 : Switch Projects
             3 : Create New Project
             4 : Discard Project
             5 : Exit to Operating System }
           case selection of
           1 : if project_id=0
               then input_error('No current project selected!  ')
               else next_module:=get_next_module(completion);
           2 : begin
                project_id:=select_project(user_id);
                if project_id=0
                then input_error('You have no Projects defined! ')
                else begin
                        completion:=get_state(project_id);
                        get_name(project_id,name);
                     end;
               end;
           3 : create_project(user_id, project_id, name);
           4 : begin
                  kill_project(user_id, project_id);
                  name:='No current project selected.  ';
               end;
           5 : begin
                  next_module:=os;
                  project_id:=0;
                  user_id:=0;
               end;
           end;
        until next_module<>undefined;
     end; { select_option }

{*****************************************************************}
{* Procedure : Execute_Next                                      *}
{* Parameters : Module, Project                                  *}
{* Global Variables : Current                                    *}
{* Entry Conditions : Module and Project will identify the       *}
{* module which is to be executed next.                          *}
{* Exit Conditions : The selected module will be executed        *}
{* after the Argument file is set up with the names of all       *}
{* of the files required by the module.  The Directory will      *}
{* have been updated to reflect any changes in the status of     *}
{* all the files used by the command processor.                  *}
{* Note that control will NOT return to the caller!              *}
```

Command Processor Module

```
{*****************************************************************}
procedure execute_next(module : modules; project : integer);
{ See the Global Directory Module for details on the following }
   procedure updategd; external;
   procedure exec(module : modules); external;

{ See the Argument Module for information on the following }
   procedure update_header(info : arg_header); external;
   procedure load_args(module : modules; project : integer);
                        external;

   begin
      update_header(current);
      load_args(module,project);
      updategd; { Update Global Directory and Close all files }
      exec(module); { No direct return from this procedure! }
   end;

{ See the Argument Module for details on the following }
procedure read_args(var info : arg_header); external;

begin { command processor }
   init_all;
   read_args(current);
   with current do
   repeat
      if error_code<>0
         then resolve(error_code);
      if user_id=0 and error_code=0
         then get_identity(user_id,valid)
         else valid:=true;
      if (valid and (error_code=0))
         then select_option(user_id, project_id, completion, module);
      if error_code=0
         then execute_next(module, project_id);
   until error_code=0
end.
```

APPENDIX F - Command Processor User's Manual

Start-up

The following discussion assumes that the computer is on and that the Command Processor has been properly installed. (See Appendix C for installation instructions.) To initiate the program, insert the Command Processor disk (it should be labeled as disk number one) into Drive A and type ^C. Next type 'CP' and press return. The Screen will clear and the following will be displayed:

```
Who Are You ? _____________________________
Response must be between 1 and 30 characters
```

In response to this question, you may enter whatever character string you want to. Your answer will be used to identify you in the future, so some form of your name seems the most likely candidate. Once you successfully answer this question, you will be asked some more questions with equally obvious responses. Rather than trying to explain here what your answers should be, I will explain how to get help and how to edit your responses. This system has been designed to not need a manual for operation, after all.

How to Get Help

Whenever you are asked a question, you have the opportunity to get an explaination of how you should respond. The question mark on the keyboard is reserved for this purpose, and cannot be a part of any response. When you type a '?', the corresponding help message will be displayed. If the message is larger than the space available on the screen, you will be prompted to press any key to see

the rest of the message.

If you press '?' while in the middle of an answer, then when you have finished reading the help message your partial answer will be erased. Don't forget and try to use a '?' as part of any answer!

In addition to help for questions, there is also help available for each menu option. When a menu is displayed on the screen, pressing '?' will retrieve the help message for the item indicated by the current position of the cursor (see below for instructions on moving the cursor). Before you select an item for the first time, you should ask for help so that the implications of that particular selection may be explained to you.

Editing Your Responses

There are three different kinds of queries to which you may be asked to respond, and each has unique editing features available. The three types are string queries, number queries, and yes/no queries.

String Queries

As you may expect, the response to a string query is an ASCII character string. Any printable character EXCEPT '?' is allowed in the string. The general format of a string query is as follows:

```
Question being asked ? _____________________________
Response must be between min and max characters
```

Min and max are numbers which indicate the allowed length of the response. The number of underlines will be the same as max.

Answer the question by typing an appropriate response. You may move the cursor back and forth with the left and right arrows. Backspacing over what you have already typed will not erase it, so you may correct a mistake at the beginning of a line without retyping the whole thing.

If you want to insert a character or characters in the middle of something you have already typed, move the cursor to where you want some space opened up, then type ^S (inSert) for each character to be inserted. Each character from the one under the cursor to the end of the string will be moved to the right every time ^S is typed. The cursor position will not change. If what you have typed is already the maximum length, the last character will be lost with every insertion.

Deleting characters is also simple - position the cursor on the character to be deleted and type ^D (Delete). All the characters to the right of the cursor will move left to fill in the space left by the deleted character. Deletion

also does not alter the cursor position.

When you are satisfied with your answer, press RETURN to terminate entry. If you have entered less than the minimum number of characters, the RETURN will be ignored. The cursor position at the time you press RETURN has no effect on the answer - all the characters you see on the screen will be in it, so be careful.

Number Queries

A number query can be recognized by the following format:

Question Being Asked ? _____ units
Range: min to max

Units will be one of mils, inches, mm, or blank. If blank, then the number is dimensionless. Min and max will be in terms of the displayed units.

The only characters recognized are the digits 0 through 9, the comma, the period, the plus sign, and the minus sign. Any other character typed will be ignored. The left and right arrows are also recognized as with the string query. Insertion and deletion are the same as for the string query, but only one period (decimal point) is allowed.

An additional feature of the number query is the ability to change units. Pressing ^U (Units) will cycle the units displayed, and also change the values of min and max appropriately. Every time ^U is pressed, the units will change in the following sequence: mils to inches to mm back to mils. If no units are displayed, pressing ^U will do nothing.

Pressing either plus or minus will change the sign of the number as indicated by the leftmost character, but the cursor will not move. Commas may be placed wherever desired, but they have no effect on the value. As indicated above, any decimal point after the first will be ignored.

Termination of a number entry is indicated by pressing RETURN. If the value does not fall within the specified range, you will be notified and asked to try again. As with the string query, the position of the cursor when RETURN is pressed does not matter. RETURN by itself will result in a zero value.

Yes/No Queries

A Yes or No query can be recognized by the format:

```
Question Being Asked ?
Please Respond with Yes or No.
```

Pressing the 'Y' key (either upper or lower-case) will spell out the word "Yes" and the 'N' key will spell out "No". Any other key will spell out "I Don't Know!". When the proper response is displayed, pressing RETURN will enter your answer. No editing is necessary, and you may change your mind as often as you like before you press RETURN.

Menu Item Selection

When a menu is displayed on the screen, you are supposed to select one of the items. A menu looks like this:

```
* Item One
  Item Two
  Item Three
  ......
  Last Item
```

The asterisk is the cursor which indicates the current item. Pressing the up and down arrows will move the cursor up and down as desired. To select an item, position the cursor beside it and press RETURN. If you're not sure what will happen with a particular item, position the cursor next to it and press '?' as discussed above. This will tell you all you need to know about the item in question.

Floppy Disk Handling

Because the system keeps track of all disks with an index number, it is important to properly label a diskette when instructed to do so. It is also important to not change disks unless specifically asked. Following these rules will insure that your files don't get corrupted.

Once a diskette is assigned to you, it will never be made available to anyone else, even if you delete the project on it. This allows you to keep your own disks physically separate from everyone else's if you wish.

Before you create a new project, be sure you have some blank FORMATTED disks available. The disk doesn't have to be entirely blank, but it must have enough room for all the project files. Just to play it safe, you should dedicate

empty disks to each project.

VITA

Ernest William Krausman was born on 17 October 1958 at Wheelus AFB, Libya. He graduated from high scrool in Daytona Beach, Florida in 1975 and attended the Georgia Institute of Technology from which he recieved the degree of Bachelor of Electrical Engineering in June 1979. Upon graduation, he recieved a commision in the USAF through the ROTC program, and entered active duty in August 1979 at Wright Patterson AFB as a foreign telecommunications systems analyst for the Foreign Technology Division, where he stayed until entering the School of Engineering, Air Force Institute of Technology, in June 1982.

Permanent address: 799 Marvin Road

Ormond Beach, Florida 32074

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/EE/83D-35	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering / 6b. OFFICE SYMBOL (If applicable): AFIT/EN	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): WPAFB, OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION / 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): Printed Circuit Board Layout by Microcomputer	
12. PERSONAL AUTHOR(S): Krausman, Ernest William	
13a. TYPE OF REPORT: MS Thesis / 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): 83 December / 15. PAGE COUNT: 272
16. SUPPLEMENTARY NOTATION	Approved for public release: IAW AFR 190-17. LYNN E. WOLAVER, Dean for Research and Professional Development, Air Force Institute of Technology (ATC), Wright-Patterson AFB OH 45433 — 7 Feb 84
17. COSATI CODES — FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): printed circuit board routing, computer aided design, interactive graphics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

TITLE: Printed Circuit Board Layout by MicroComputer
ADVISOR: Lt. Col. Hal Carter

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED [X] SAME AS RPT. [ ] DTIC USERS [ ]	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Lt. Col. Hal Carter	22b. TELEPHONE NUMBER (Include Area Code) / 22c. OFFICE SYMBOL: AFIT/EN

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

19. Abstract
Printed circuit board artwork is usually prepared manually because of the unavailability of Computer Aided Design tools. This thesis presents the design of a microcomputer based printed circuit board layout system that is easy to use and cheap. Automatic routing and component placement routines will significantly speed up the process.
The design satisfies the following requirements: Microcomputer implementation, portable, algorithm independent, interactive, and user friendly. When fully implemented, a user will be able to select components and a board outline from an automated catalog, enter a schematic diagram, position the components on the board, and completely route the board from a single graphics terminal.
Currently, the user interface and the outer level command processer have been implemented in Pascal. Future versions will be written in C for better portability.


FILME
4-84